DRA

SOFTWARE ENGINEERING LABORATORY SERIES SEL-80-002

MULTI-LEVEL EXPRESSION DESIGN LANGUAGE—REQUIREMENT LEVEL (MEDL-R) SYSTEM EVALUATION

MAY 1980

NASA
National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES SEL-80-002

MULTI-LEVEL EXPRESSION DESIGN LANGUAGE—REQUIREMENT LEVEL (MEDL-R) SYSTEM EVALUATION

MAY 1980

NASA

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC (Systems Development and Analysis Branch)
The University of Maryland (Computer Sciences Department)
Computer Sciences Corporation (Flight Systems Operation)

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document. A version of this document was also issued as Computer Sciences Corporation document CSC/TM-80/6093.

The primary contributors to this document include

William Decker	(Computer Sciences Corporation)
Charles Goorevich	(Computer Sciences Corporation)

Other contributors include

Arthur Green	(Computer Sciences Corporation)
Frank McGarry	(Goddard Space Flight Center)

Single copies of this document can be obtained by writing to

Frank E. McGarry
Code 582.1
NASA/GSFC
Greenbelt, Maryland 20771

ORIGINAL PAGE IS
OF POOR QUALITY

ABSTRACT

An evaluation of the Multi-Level Expression Design Language - Requirements Level (MEDL-R) system was conducted to determine whether it would be of use in the Goddard Space Flight Center Code 580 software development environment. The evaluation is based upon a study of the MEDL-R concept of requirement languages, the functions performed by MEDL-R, and the MEDL-R language syntax. Recommendations are made for changes to MEDL-R that would make it useful in the Code 580 environment. This document has been prepared in partial fulfillment of the requirements of Task 990 of National Aeronautics and Space Administration contract NAS 5-24300.

ORIGINAL PAGE IS
OF POOR QUALITY

TABLE OF CONTENTS

ORIGINAL PAGE IS
OF POOR QUALITY

TABLE OF CONTENTS (Cont'd)

ORIGINAL PAGE IS
OF POOR QUALITY

TABLE OF CONTENTS (Cont'd)

ORIGINAL PAGE IS
OF POOR QUALITY

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1 - INTRODUCTION TO REQUIREMENTS LANGUAGES

The lack of proven techniques for specifying and performing analysis on requirements causes many serious problems in the development of software systems. To fill this void, a class of system development tools known as Requirements Analysis Languages (RALs) has become available. Each RAL has its own approach to the problem of specifying and performing analysis on requirements. This fact is not surprising, since no standard format exists in which requirements are given, and requirements analysis is a relatively new concept.

This document presents a summary of work done in evaluating the Multi-Level Expression Design Language - Requirements Level (MEDL-R), a RAL that is part of the Multi-Level Expression Design System (MEDSYS) (References 1 and 2). The remainder of this section contains further definition of what a RAL is and how RALs fit into the software development cycle. The Goddard Space Flight Center (GSFC) Code 580 environment is defined, and the criteria for judging a RAL in this environment are presented. The conclusion of this section is an overview of two different approaches to the problem of specifying and performing analysis on requirements; one of these is the MEDL-R approach. Section 2 contains a detailed analysis of the functions and structure of MEDL-R, including the specific strengths and weaknesses of the current system. Section 3 contains an analysis of the current MEDL-R requirements language syntax and some recommendations for enhancements. Section 4 contains general conclusions on the system and its applicability to the Code 580 environment.

1.1 GOALS IN USING RALs

RALs are intended to assist the developer in the creation of a rigorous statement of a system. The developer needs this automated help because today's systems are too large and too complex to be developed effectively using manual methods. The amount of material that must be covered in a system specification often leads to errors due to

- Omissions
- Ambiguities
- Inconsistencies and contradictions
- Lack of clarity and precision
- Varying levels of detail
- Presence of design type constraints

The cause and effect of each type of error are explained below.

Omissions are either intentional or accidental. Intentional omissions are caused by differing rates in developing the definition of parts of a system. For example, if the specifications for one subsystem are not available, the specifications for other subsystems may intentionally omit the specifications for interfacing with the missing subsystem. It is up to the developer either to stop and allow the schedules to match or to proceed. In either case, intentional omissions must be made highly visible to everyone involved in the specification effort. Accidental omissions typically are caused by the large volume of material or by a lack of communication between personnel.

An omission (either intentional or accidental) that is encountered in the design phase results in either an arbitrary decision by the design team or a delay while the specifications are completed.

RALs assist in the detection of omissions in several ways. First, the format used to enter the requirements into the system creates a regimented environment. This environment encourages rigorous and logical organization of the material before entry. Second, the data base created and maintained by the RAL is always available in its current form. This

ORIGINAL PAGE IS
OF POOR QUALITY

encourages constant reevaluation of the state of completion of the system and allows project management to concentrate effort in the areas where weaknesses are apparent. The RAL assists in reducing the volume of material through categorization or subdivision of the system. Communication between personnel is increased because each person has access to the current system description. If an omission is detected, it may be flagged and immediately brought to the attention of all concerned.

Ambiguities occur because the English language is used to express the original statement of the system. The interpretation of each English statement is dependent upon the individuals making the statement or reading it.

Ambiguities result in the development of a system that does not match the original intent of the specifications. As with omissions, ambiguities result in (1) delays while waiting for clarification, (2) arbitrary interpretations made in the design phase, or (3) incorrect designs.

RALs help with the detection of ambiguities through either the enforced adherence to a rigid description syntax in which the meanings of terms are fixed or the establishment of fixed review policies as part of the "analysis" of the requirements.

Inconsistencies and contradictions result from poor communication between personnel, as well as from ambiguities and omissions. The large volume of description required for even a moderate system tends to isolate each individual involved from the work done by others and thus prevents the early detection of the causes.

Inconsistencies and contradictions usually result in the scrapping of part of the system. The inefficiency of discarding completed work results in cost overruns, schedule delays, or even failure to implement the complete system. When they are detected, contradictions must be treated as symptoms and traced back to locate the root cause.

RALs can be of aid by helping to eliminate the ambiguities or omissions that cause inconsistencies and by providing a tracing mechanism to quickly locate the source of the problem if one is detected.

Lack of clarity and precision and varying levels of detail are actually two sides of the same coin. Many individuals tend to be overly thorough in describing a known subject and to gloss over the description of an unfamiliar topic. The result is masses of explicit detail that obscure the fact that some portions of the system are not adequately described. The failure to completely describe the system may become apparent as late as the system acceptance testing phase.

RALs can help here in two ways. First, RALs may organize the specification in a hierarchical (top-down) format; then, with proper presentation tools, the depth of knowledge on any given specification may be examined and the even progress of development across the entire system ensured. Second, RALs may use an approach that embodies the idea that each statement about the system must be testable in the final system and also encourages the test to be included with the statement. This virtually eliminates the possibility that the description lacks precision.

The presence of design type constraints results from a natural tendency to solve problems. Most individuals who work on system specifications have "graduated" from system design. The individual's design skill, when combined with a difficulty in verbalizing the problem, often results in a design solution of the problem instead of a clear statement of the problem to be solved.

This situation may result in a series of solutions that are not based upon a conception of the entire system. When the system is finally brought together, conflicts may arise that could have been avoided if the constraints imposed by the "solutions" were not present. Thus, although perhaps simplistic, in some environments the following statement is a good rule of thumb: Requirement specification states what a system will do; design specification states how the system will do it.

RALs can help minimize this difficulty by providing each specification for a system with an optional trial design solution. This serves as a constant reminder that the solution is not the problem.

1.2 USE OF RAL IN CODE 580 ENVIRONMENT

Section 1.1 describes the purposes and goals of RALs in general terms. To obtain a meaningful evaluation of a particular software tool, the tool must be measured against the environment in which it will be used. Section 1.2.1 describes the GSFC Code 580 software development environment in terms of the current procedures used and the types of software systems developed. Section 1.2.2 presents a set of criteria (drawn from the description of that environment) against which each potential RAL must be measured. The reader is cautioned that the remainder of this document contains opinions, comments, recommendations, and judgments based on the contents of this section. Other environments will demand other sets of criteria that would naturally affect any evaluation. This evaluation is strictly aimed at determining the suitability of MEDL-R in the Code 580 environment.

ORIGINAL PAGE IS
OF POOR QUALITY

1.2.1 GSFC CODE 580 SOFTWARE DEVELOPMENT ENVIRONMENT

GSFC Code 580 is responsible for software development in the following areas:

- Spacecraft attitude determination and control
- Spacecraft orbit determination and control
- Spacecraft maneuver planning
- Mission planning

In Code 580, the software development cycle starts with a contract for the preparation of a formal requirements specification document. The organization preparing this document is outside of GSFC and is not necessarily the organization contracted to perform the design, implementation, and integration and testing of the system. The delivered document is expected to contain detailed functional (not procedural) specifications for the system. The document is prepared by individuals with approximately 4 to 5 years of experience in flight dynamics and celestial mechanics applications. The document is typically delivered within 12 months of the start of the contract. The expenditures for this phase of the software development cycle are about 25 percent of the total for a particular system if the system is significantly different from previous systems. The percentage expenditure is typically reduced when (as is often the case) the system is similar to previously developed systems. Recently, Code 580 has initiated new development efforts at a rate of two to three per year.

The second portion of the software development cycle starts with a contract for the design, implementation, and integration and testing of the system. The organization performing these functions starts with an analysis of the requirement specifications document. Often the system is similar to previous systems, resulting in the extensive reuse of code, personnel experience and skills, and documentation. A typical system spends 12 to 20 months in design, implementation, and integration and testing. The phases are not easily separable and often overlap in an iterative way. This results from a large number of changes to the functional specifications. Typically, for the first 75 percent of the contract, two to three nontrivial revisions occur per week. Delivery dates for fully documented, operational systems are fixed by spacecraft launch schedules. Code 580 has no control over these schedules.

The management of the software development efforts in Code 580 is characterized by two levels. GSFC provides general coordination of the effort. Communication between contractors, progress monitoring, and resolution of schedule modifications are all directed by GSFC. Contractors provide tight management of the detailed technical efforts in their support areas.

Code 580 proposes to use a RAL at the beginning of the design phase to assist in the analysis of the requirements specification document (Figure 1-1). The results of the analysis will be passed to the design team to assist it in coordinating changes in the design work in response to modifications to the functional specifications.

The use of a RAL in the Code 580 environment is a natural extension of previous efforts in which other software engineering tools and methodologies have been introduced and evaluated to determine the benefits to be derived from their use. Tools such as the automated Process Design Language (PDL) processor and the Structured FORTRAN (SFORT) processor have been evaluated and found to be beneficial in the design and implementation phases, respectively. Code 580 is now investigating tools to ease the transition from requirements to design.

ORIGINAL PAGE IS
OF POOR QUALITY

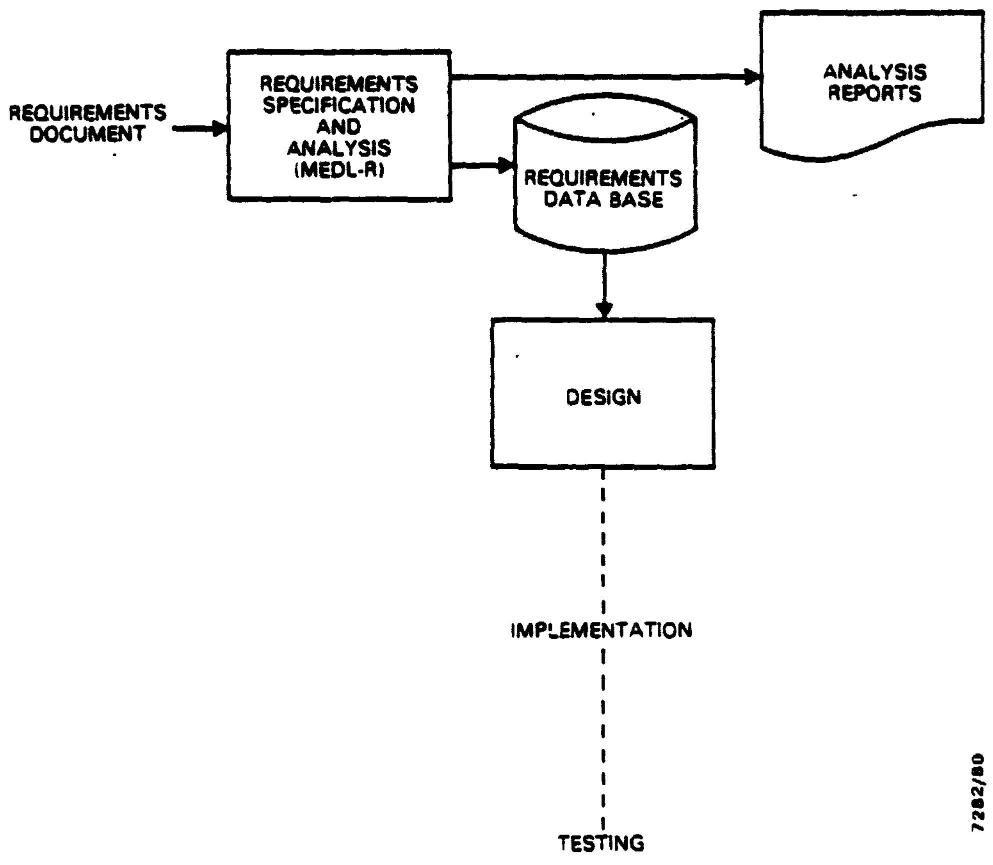

Figure 1-1. Placement of MEDL-R in GSFC Software Development Cycle

Code 580 software development results in software systems that range in size from 5,000 to 120,000 lines of code. A typical (average) system has 40,000 lines. When possible, a high-order language (typically FORTRAN) is used. The development is done on both PDP-11/70 and IBM S/360 computers.

The software can be characterized as scientific application systems with little or no real-time or near-real-time requirements. Attitude determination and control systems require software to access large data bases and to perform flight dynamics analysis. Orbit determination and control systems require celestial mechanics software that is mainly mathematical and algorithmic. Spacecraft maneuver planning requires mathematical and algorithmic software that models a particular vehicle's physical and dynamic characteristics. Mission planning software is the generalized maneuver planning software that is used to evaluate vehicle performance while the total mission is still in its definition phase.

1.2.2 CRITERIA TO BE APPLIED TO EVALUATION OF MEDL-R IN CODE 580 ENVIRONMENT

The following criteria, based upon the needs, resources, and goals of Code 580, will be used to evaluate MEDL-R:

- The system must operate efficiently within the resources of the PDP-11/70 computer allocated for use by the Software Engineering Laboratory (SEL). The PDP-11/70 is an interactive, nonbatch facility with limited printer capability.
- The system must assist Code 580 in performing an analysis of a requirement specification document with minimal impact on the schedules for other development work. The task of translating the document into a form suitable for system analysis should

be straightforward and should not change the character of the specifications; that is, they should remain "functional specifications."

- The system must be easy to learn and use effectively. Personnel without previous experience with the system, but with experience with requirements analysis, should be able to make effective use of the system. The user/system interface must be implemented in a conversational and instructive way.
- The system must be adaptable. After the system has been evaluated through actual use, the recommended changes must be easy to implement. GSFC anticipates that any requirements analysis language will require some "tuning" to fit its needs, especially in the areas of the content of analysis reports and the terms used in the language.
- The system must allow changes to the requirements and provide analysis in which the implications of the changes are presented. GSFC requires a tool that has dynamic as well as static analysis of the requirements.

1.3 EXAMPLES OF DIFFERENT RALs

The evaluation of MEDL-R required an examination of other similar software tools. To supply the background against which the MEDL-R evaluation was performed, this section discusses several approaches used to implement RALs. Examples of available RALs are the Problem Statement Language/Problem Statement Analyzer (PSL/PSA) (also known as the User Requirements Language/User Requirements Analyzer (URL/URA)), the Input/Output Requirements Language (IORL), and the Requirements Statement Language/Requirements Engineering and Validation System (RSL/REVS). Each RAL aids in the translation of a requirements document (a highly abstract, conceptual definition of a potential system) into a complete set of concrete and correct statements describing the system. Section 1.3.1 describes the PSL/PSA approach to this problem, and Section 1.3.2 describes the MEDL-R approach. Section 1.3.3 briefly summarizes the comparison.

1.3.1 PSL/PSA APPROACH TO REQUIREMENTS SPECIFICATION/ ANALYSIS

The approach used by the majority of RALs available today, including PSL/PSA (URL/URA), IORL, and RSL/REVS, differs from the MEDL-R approach. PSL/PSA is used as the example in this discussion.

The PSL/PSA system is composed of two parts: a syntactically rigorous language (PSL) and a language analyzer (PSA). PSL/PSA describes a system with a language composed of "system elements" (e.g., INTERFACE, INPUT, OUTPUT, SET, GROUP, PROCESS, GENERATE). The interrelationships between elements are rigorously defined. The developer describes the system in a language using these elements either during the specification process or after a specification document has been prepared. In either case, the developer performs a translation from text to language elements using a rigid format. The system elements chosen are terms familiar to system designers.

An advantage of this approach is that the language elements have been selected such that they (and their relationships) can be subjected to a rigorous automated analysis by the language processor (PSA). For example, a defined OUTPUT for which there is no PROCESS to GENERATE it can be easily detected by PSL/PSA. In addition, the choice of language terms very familiar to system designers results in a final system definition that is easily interpreted in the next system development phase (system design).

ORIGINAL PAGE IS
OF POOR QUALITY

PSL/PSA (and the other similar RALs) attacks the problems of omission, ambiguity, inconsistency, and contradiction through the use of a rigid syntax. Omissions are detected because the number of language elements is limited, and each relationship is rigorously defined. As with the previous example, gaps in the specification are readily apparent to the processing parts of PSL/PSA. Ambiguity is eliminated by fixing the definition of each language element. Once the developer assigns a component of the system to a class of elements, ambiguity is removed. Another individual may then examine the element class and know the properties of the system component. Inconsistencies and contradictions are easily detected because interrelations are rigorously defined. The syntax used does not allow multiple definitions of any system component. Only in the case in which subsystems are specified separately and then combined into the total system does a problem with contradictions exist, and even then the RAL detects the problem immediately upon integration.

PSL/PSA is probably less efficient in eliminating problems of lack of clarity and precision and varying level of detail. It includes no provision for supplying test criteria to ensure that the final specification is a clear and precise statement of the system. In addition, it provides no ground rules for expanding the definition of the system, a lack that may lead to overspecifying one portion and leaving another portion in a virtually undefined form.

Perhaps the weakest aspect of the PSL/PSA approach, when compared to the previously stated general goals of a RAL, is in the elimination of design type constraints. The following three critical comments can be made about this particular

ORIGINAL PAGE IS
OF POOR QUALITY

implementation in regard to the presence of design type constraints:

1. Due to the translation that takes place, contact may be lost with the original requirement statement. The result of translating the original statement into language elements is requirements source code that does not resemble the original text. Because of this translation, the resulting specifications may not be comprehensible to the originator. The output of this type of RAL is, as previously noted, in a form easily understood by a designer, but this eliminates effective feedback to the originator. Part of the clarification process involves the validation of the rigorous statement of the system by the originator (who should be able to verify the final set of requirements directly in the terms used in the original specification). An optional DESCRIPTION element in PSL/PSA gives the analyst the opportunity to enter the original text; but if the analyst elects not to do so, the resulting specification has no direct references.

2. The language itself may influence the contents of the data base and thus the final design. The rigid structure of the language may result in a restricted statement of the system requirements, thus removing a certain amount of flexibility from the design stage. If the syntax of the RAL is not capable of describing a particular type of solution system, that system is eliminated from consideration. The elimination of possible solutions should be part of the design stage and not part of requirements specification.

3. The potential exists for actually designing the solution system with this type of RAL. The language is composed of "design terms," and this encourages the emphasis of detail. These features can lead to the explicit introduction

of design type constraints, which are not desirable at this (the requirements specification/analysis) stage.

Code 580 has evaluated PSL/PSA in its environment and has reached the following conclusions:

- The PSL/PSA system is too large. A scaled-down version (URL/URA) was tested on the IBM S/360 computer at GSFC. The decision not to attempt to further reduce the program to fit within the resources of the PDP-11/70 was made because of the complexity of the system (a quality not unusual in an older, established system). The complexity would have caused difficulties in the implementation of system extensions that Code 580 found desirable.
- The language syntax was not appropriate or particularly easy to use in the specific application of analyzing a Code 580 requirement specification document. PSL/PSA appears to be more of a design tool than a requirements analysis tool when it is applied to this application in this environment.

1.3.2 MEDL-R APPROACH TO REQUIREMENTS SPECIFICATION/ ANALYSIS

In the MEDL-R approach, the original statement of a system requirement is entered into a computer data base in its English-language form. No translation of the statement is performed. The statement becomes the central item of each MEDL-R "requirement." Instead of translation, the developer adds to the requirement by supplying qualifiers (and in some cases, quantifiers) that characterize the meaning of the English text. The relationships between requirements may also be specified; for example, a requirement may be "derived from" other requirements.

ORIGINAL PAGE IS
OF POOR QUALITY

An advantage of the MEDL-R concept of requirement analysis is the retention of the original statement of the system. If one particular requirement needs supporting statements to clarify it, the supporting statements are entered as requirements themselves and are flagged as having been derived from the original. This procedure allows the total information concerning the system to grow without modification to the original specification.

Unlike the PSL/PSA translated requirement, the MEDL-R text format is not easily subjected to automated analysis, but the central position of the text within the requirement helps to ensure that the original intent of the specification is preserved and brought to the attention of the developer at every opportunity. The MEDL-R system is intended to capture requirements and support management control and traceability of requirements.

MEDL-R has few facilities for detecting errors of omission, ambiguity, inconsistency, and contradiction. These types of errors must be located and corrected by the user. Omissions might be detected by an examination of the number of statements in each NATURE category (see Section 3.1.3), on the assumption that each of the categories must be addressed in some detail in order to adequately describe a system. This assumption is warranted only to the extent that the given categories are valid and exhaustive for the type of system being described. Ambiguities are quite likely to occur in the MEDL-R requirements because the requirements are retained in their English-language form. The EXPLANATION entry (see Section 3.1.13) of each requirement can be used to note a possible clarification for each questionable statement. Inconsistencies and contradictions can be detected by suitable combinations of meaningful SUBJECT categories (see Section 3.1.12) and data base QUERYs (see Section 2.1.4).

ORIGINAL PAGE IS
OF POOR QUALITY

MEDL-R could be easily modified to help eliminate errors of lack of clarity and precision and varying level of detail. TEST-CRITERIA could be added as a new entry in a MEDL-R requirement, thus linking the specification and its validation procedure. The hierarchical structure of MEDL-R requires further implementation to give the system a means for checking on the level of detail specified.

MEDL-R excels in eliminating design type constraints. The requirement specifications are present in the data base in the form in which they are given by the originator. The logical development of the full system description is derived from these in a language understandable by the originator. The addition of a new entry (REVIEWED-BY) would signify that the originator had read, understood, and agreed with each requirement in the data base.

1.3.3 COMPARISON SUMMARY

PSL/PSA is composed of a language and a language analyzer. This RAL's strong points are its ability to detect ambiguity, omissions, and inconsistencies. It is based on concepts familiar to system designers. It is a mature, complex tool that is representative of the mainstream implementation of RALs.

MEDL-R uses a concept of requirement categorization and an analyzer to present relationships among requirements by category. Its strong points are its adaptability to a hierarchical (top-down) evolution of specification and the retention of the original specification of the system in terms understandable to the originator. It is a new concept in RALs and has not yet reached its maturity. Its implementation is still in a formative stage and may be modified with relatively small effort once its failings have been identified.

ORIGINAL PAGE IS
OF POOR QUALITY

SECTION 2 - MEDL-R ANALYSIS

This section provides an analysis of the current MEDL-R system. The seven major MEDL-R subsystems are examined as to their applicability, completeness, and structure. The strengths and weaknesses of each subsystem are specified, and, when appropriate, recommendations are made to improve weaknesses. Section 2.1 discusses the functions performed by each MEDL-R subsystem, including output reporting, data base editing, and analysis capabilities. Section 2.2 provides flow diagrams of user/system interaction. Section 2.3 describes the system structure and provides an overview of the relationship between functions and structure. Section 2.4 specifies a recommended structure and data flow for the MEDL-R system based on the comments in Sections 2.1 through 2.3.

2.1 FUNCTIONAL ANALYSIS

The MEDL-R system comprises seven major subsystems:

- Create (CRE)--allows the entering of initial requirements into the system
- Update (UPD)--allows the updating of existing requirements or the adding of new requirements
- Language Translator (LTR)--allows the movement of both initial requirements and updated requirements into the MEDL-R requirements data base
- Query (QRY)--allows the user to extract summary data from the data base based on requirement descriptors and their arguments
- Analyzer (ALZ)--supports predefined analyses of the data base; currently develops reports on the requirements data base

- Metric (MET)--provides analytic measures of the data base
- Change System Name (CSN)--provides the user with access to another MEDL-R data base without exiting the system

The following subsections present a detailed description of each program subsystem, specify the strengths and weaknesses of each, and provide general recommendations for correcting the weaknesses.

2.1.1 CREATE SUBSYSTEM

The Create subsystem is used to generate the initial requirements source file from a user terminal. Entry is via a category-by-category prompt (e.g., ENTER DESCRIPTION OF REQUIREMENT >). If the source file already exists when the Create subsystem is activated, the new requirements are appended to the existing source file. To avoid unnecessary prompts, a special "prompt limit" feature exists to control the amount of data the system will request. Special symbols are used for various purposes: proper line termination (. or ..), lists of entries (entries are separated by commas), nonrequired entry (0 or blank), comment entries (#), and session termination ($). Limited error checking is done on the user file at the time of entry. If the input source file is created in the proper format from card input or by the system text editor, the Create subsystem need not be exercised. Create subsystem output is used as input to the Language Translator subsystem.

The strengths of the Create subsystem are as follows:

- It provides clear, precise prompts for information.
- It can specify the prompt limit.

The weaknesses of this subsystem are as follows:

- It may not be invoked after the first invocation of the Language Translator subsystem.
- The use of a variety of control symbols in this input editor is confusing to the general user, especially since this subsystem is essentially used only once.
- It performs some checking of invalid keywords and input errors, a function that should be the job of the Language Translator subsystem.

General comments regarding the Create subsystem are as follows:

1. Some errors and inefficiencies have been identified in the Create subsystem. These are mostly associated with reinvoking the subsystem before passing the output data set to the Language Translator subsystem.
2. The input data set to the Language Translator subsystems may be a card deck or a data set created by the systom text editor.
3. The Create subsystem is still necessary (and perhaps preferable) since its use does not require direct knowledge of the format of the input data set.

The idea of specialized editor from which the user can enter his/her input requirements quickly and with a few basic commands is a good one. However, to be effective, this same editor must be available at all phases of the requirements analysis. The input procedure for a requirement must always appear the same to the user regardless of whether the requirement is part of the initial set, is added later during data base expansion, or is being updated.

2.1.2 UPDATE SUBSYSTEM

The Update subsystem is available to the user when all requirements data base files for the current system have been specified (currently, six files make up the requirements data base). The subsystem allows the user to correct or add to the existing system. Update operations include changing any user-entered field of an existing requirement, deleting or adding entries within an existing requirement, changing a requirement name or system name, and adding all new requirements to the existing system. Like Create subsystem output, Update subsystem output is used as input to the Language Translator subsystem. Unlike the Create subsystem, the Update subsystem requires as input an existing requirements data base built by the Language Translator subsystem.

The Update subsystem allows the user to (1) list the update file on a display terminal, (2) change a requirement name, and (3) enter a new requirement. The Update subsystem has its own complete set of rules for making changes to requirements. These rules are different from those of the Create subsystem.

The strengths of the Update subsystem are as follows:

- Updates can be made quickly.
- No syntax checking is performed.

The weakness of this subsystem is that its use requires direct knowledge of where input information is to be placed.

The general comments made in Section 2.1.1 concerning the Create subsystem also apply to the Update subsystem. In addition, as stated in the same subsection, the method of input, whether the input is a new requirement or an update to an existing requirement, must appear the same to the user.

ORIGINAL PAGE IS
OF POOR QUALITY

2.1.3 LANGUAGE TRANSLATOR

Using input from either the Create subsystem or the Update subsystem, the Language Translator subsystem builds or modifies the requirements data base. Once the data base is built or updated, the input source (either original or updates) is no longer needed. Once started, the translation and data base build or update process is completely automatic and requires no user intervention. While performing this task, the Language Translator subsystem produces a source input listing file and flags any errors detected. The user must use system utilities to purge previous versions of the listing file. When the translation is complete, a message is sent to the user indicating so and specifying the number of errors detected. If no unrecoverable errors were detected, the input source (original or updates) is deleted from the system. When modifying the requirements data base, the same PDP-11/70 Files-11 version is retained.

The strengths of the Language Translator are as follows:

- It is the only subsystem that modifies the data base. Thus, as new error checks and more complicated language translations are identified, only the modules in this subsystem need be expanded.
- It organizes the requirements into one complete set, which can then be listed by the Analyzer subsystem.

The weaknesses of the Language Translator subsystem are as follows:

- The user has no direct connection to the six files comprised in the requirements data base; merely listing them does not present the requirements to the user in a clear, precise statement. Provisions are made to dump the contents of the current data base files as well as to provide listings files on disk. In both cases a separate offline utility

ORIGINAL PAGE IS
OF POOR QUALITY

program must be run to view the output (i.e., program SUMREL for a dump of the requirements data base and systems utility programs for a display of the translators listing file). Normally, to look at the data base in detail, the user enters the Analyzer subsystem. At the translation point, the user is usually interested only in requirements that have been modified or just entered into the system (i.e., interested in ensuring that they were received properly). Thus, software to examine only new or modified requirements immediately after translation would be useful.

- The deletion of the input data by the system is not optional. To eliminate the possibility of losing information in the event of a system crash or a large number of recoverable errors, the deletion of the input files should be a user option.

2.1.4 QUERY SUBSYSTEM

The Query subsystem allows the user to search the requirements data base for items that have text in common. The QUERY command has the general form

$$< \text{statement-type} > \; = [< \text{string} > [\{\begin{smallmatrix}\text{and}\\ \text{or}\end{smallmatrix}\} < \text{string} >]]$$

where [ ] indicates optional parts of the command, { } indicates that a selection is to be made, < string > is an arbitrary character string of less than 30 characters, and < statement-type > may be any of the following: NATURE, VERSION, MOTIVATION, SUBJECT, STATUS, DESCRIPTION, FUNCTION-RESOLUTION, DATA-RESOLUTION, RESOURCE-RESOLUTION, or SUBSYSTEM-RESPONSIBILITY. The following are examples:

```
QRY > SUBJECT
QRY > NATURE = DATA
QRY > SUBJECT = OPERATING-SYSTEM OR I/O
```

The output from a QUERY command is a list, displayed on the user's screen, of all requirement identifiers (IDs) that satisfy the specified condition. These requirement IDs are also used on an output file along with their DESCRIPTION and VERSION; following termination of the MEDL-R system, the user can use system utilities to list their contents on a lineprinter or cathode ray tube (CRT) terminal. There are some additional rules concerning use of the Query subsystem; however, unlike the rules of the Create subsystem, they minimally affect the user.

The strengths of the Query subsystem are as follows:

- It provides a high degree of flexibility in searching through requirements for inconsistent and ambiguous terms.
- Its "and/or" option enhances the search capability.

The weaknesses of the Query subsystem are as follows:

- It is extremely slow for large data bases.
- The QUERY command syntax is not sufficiently general to handle many types of questions about the data base. A helpful modification would be the capability to allow searching on two different statement types (e.g., QRY > SUBJECT = INPUT AND STATUS = ACTIVE).
- Like other MEDL-R subsystems, the Query subsystem creates files that the user must leave the system to examine.

ORIGINAL PAGE IS
OF POOR QUALITY

- The subsystem cannot access several MEDL-R statement types (e.g., ORIGINATOR, SOURCE, SCOPE, EXPLANATION). With modification, in certain environments or with certain types of systems, the Query subsystem could produce beneficial information about these types.

2.1.5 ANALYZER SUBSYSTEM

The Analyzer subsystem is primarily used to allow the user to examine and obtain lineprinter copies of the requirements data base. The Analyzer subsystem accepts the following commands:

- SUMMARY
- LISTNAME = < name >
- LISTALL
- FRS

The SUMMARY command produces three tables that provide statistics on the NATURE[1] keyword, the SUBJECT[1] keyword, and relation types. The LISTNAME command produces a listing of all information contained in the data base about a specified requirement ID. The LISTALL command performs the same function as LISTNAME; however, the Analyzer subsystem automatically performs a LISTNAME for every requirement ID contained in the data base. The FRS command builds the Formatted Requirements Statement file, which can be spooled to the lineprinter for an easily readable, formatted hardcopy output of the requirements in the data base. Individual requirements or all requirements can be specified for the final FRS file. (The FRS format closely resembles the format used by the Update subsystem when presenting a requirement for modification.)

[1]Language keywords are described in Section 3.

ORIGINAL PAGE IS
OF POOR QUALITY

Provisions were built into the Analyzer subsystem to perform specific analysis on COMPLETENESS, COMPLEXITY, and CONSISTENCY, but these capabilities have not yet been implemented.

The strengths of the Analyzer subsystem are as follows:

- It allows the generation of a lineprinter copy of the requirements data base, including the following information for each requirement: number of changes, origin data, last edit date, and completion date. Each requirement is formatted to fit on one standard 8-1/2-by-11-inch page, with overflow pages immediately following.
- It allows "stubs" for additional analysis.

The weaknesses of the analyzer subsystem are as follows:

- The SUMMARY report contains statistics about the entire data base, usually on a number-and-percentage basis (e.g., the number and the percentage of requirements containing NATURE keywords). The user has no control over the content of this report, and the data are automatically inserted into the Analyzer subsystem output file.
- As with the Create subsystem, some errors in the Analyzer subsystem have been identified. For example, (1) the percentages presented in the SUMMARY report are not correct, (2) the DESCRIPTION text, presented as part of the LIST and FRS output, may be truncated at the end of a line. The last is disastrous if words such as "not," "all," "some," and so forth are lost; misunderstandings also may arise when SMM-OBC becomes SMM.

ORIGINAL PAGE IS
OF POOR QUALITY

Two general comments must be made about the Analyzer subsystem:

1. The purpose of the Analyzer subsystem in its present form might be more apparent to the user if it were called the Report Generator subsystem.

2. The SUMMARY report, currently produced as an option in the Analyzer subsystem, should be expanded for the following purpose. Many entries in the standard MEDL-R requirement are in the form of strings. The Query and Metric subsystems perform operations based upon the matching of strings from one requirement with strings from other requirements. A misspelled or incomplete string will not result in a match with the "accepted" form of the string (e.g., I/O-DEVICE will not match with IO-DEVICE, I/O-DEVICES, or I/O-DEVISE). The SUBJECT summary, currently included in the SUMMARY report, provides the user with a way of detecting these near-duplicates. The user may locate and modify the SUBJECT strings (using the Query and Update subsystems) if inconsistencies are present. The suggested expansion would add optional summaries of the current strings in each of the following statement types: VERSION, SUBSYSTEM, SOURCE, CONSTRAINT, FUNCTION-RESOLUTION, DATA-RESOLUTION, RESOURCE-RESOLUTION, SCOPE, ORIGINATOR, RESPONSIBILITY, REPLACES, and DERIVED-FROM.

2.1.6 METRIC SUBSYSTEM

The Metric subsystem provides analytic measures of the data base. It differs from the Analyzer subsystem in that it does not derive its information directly from the requirements data base but instead requires a routine to transform

ORIGINAL PAGE IS
OF POOR QUALITY

the data base into a form suitable for graphic analysis. The transformed output is divided into two parts: part 1 contains the NATURE, SUBJECT, and MOTIVATION relationships;[1] part 2 contains the RESOURCE-RESOLUTION relationships. A relationship is defined between any two requirements if they have the same NATURE, SUBJECT, MOTIVATION, or FUNCTION-RESOLUTION, DATA-RESOLUTION, or RESOURCE-RESOLUTION. A count of the number of identical relationships is maintained. For an initial requirement to be transformed, it must meet the following criteria: (1) the VERSION and STATUS of the requirement match that requested by the user in the Metric subsystem; (2) the requirement is not OBSOLETE; and (3) the requirement does not contain a REPLACED-BY or DERIVES statement.

Once the requirements have been transformed, graphic analysis can be performed. Table 2-1 (the contents of which are taken from Reference 1) lists the allowable commands and the functions performed in this analysis. The analysis is very formal in that it performs accepted mathematical techniques. The results of the analysis are presented in matrix format.

Of the seven MEDL-R subsystems, the Metric subsystem is the one in which the least experience has been gained in its use and operation. This is because only parts, rather than a complete set, of requirements have been translated for analysis. Nevertheless, a few observations on this subsystem can be made:

1. The mathematics performed, although probably very powerful, are presented in the requirements document (Reference 1) in terms unfamiliar to most people doing requirements analysis. The benefit

[1]The language elements NATURE, SUBJECT, MOTIVATION, and so forth, are described in detail in Section 3.

Table 2-1. Graphic Analysis Commands

COMMAND	FUNCTION
PRELIMINARY GRAPH TRANSFORMATION	
STABLE	COMPUTE ADJACENCY MATRIX FROM INPUT GRAPH MATRIX
DISTANCE	COMPUTE DISTANCE MATRIX FROM ADJACENCY OR SEQUENCE MATRIX
RELABEL n	REARRANGE NODES; n IS NUMBER OF ITERATIONS OF REARRANGEMENT
WEIGHT	APPLY LINK WEIGHTING FACTORS
DECOMPOSITION AND EVALUATION	
CLUSTER n	DEFINE CLUSTERS; n IS CLUSTER DEFINITION METHOD (= 0, CLUSTERING ALGORITHM; ≠ 0, USER ENTRY, i.e., USER ENTERS NUMBER OF CLUSTERS IN AN I3 FORMAT FOLLOWED BY VALUES OF THESE CLUSTERS IN AN I3 FORMAT)
EVALUATE	COMPUTE STRENGTH AND COUPLNG MEASURES OF A GRAPH DECOMPOSITION
ANDREU = n	PERFORM ANDREU'S DECOMPOSITION BASED ON A SIMILARITY MATRIX; n IS ANDREU'S "p" VALUE
HOUSEKEEPING AND FILE MAINTENANCE	
PRINT	PRINT CURRENT MATRIX
STAT	COMPUTE STATISTICS ON CURRENT MATRIX
RELOAD = n	RESTORE CURRENT MATRIX FILE OF TYPE n (= 1, ADJACENCY; = 2, WEIGHTS; = 3, DISTANCE; = 4, SEQUENCE)
STORE	SAVE CURRENT MATRIX
SUBMAT	SUBMATRICIZE CURRENT MATRIX
EXIT	EXIT THIS TYPE OF RELATIONSHIP MATRIX AND RETURN TO SELECT OTHER RELATIONSHIP MATRIX OR EXIT TO RETURN TO COMMAND LANGUAGE INTERPRETER (CLI)

7282/80

to be derived by the user in going through this analysis is unclear.

2. The matrix format is awkward. In the case of a large data base (say, 100 requirements), the output appears as a series of 100-by-100 matrices, which are impossible to present clearly on a terminal or a printer.

3. This subsystem (and not the Analyzer subsystem) should contain the planned COMPLEXITY, COMPLETENESS, and CONSISTENCY functions, since most analysis will eventually require transformations.

4. As is the case with the MEDL-R system in general, this subsystem maintains many files that the user cannot interpret or has no general interest in following a run. These files should be scratch files.

2.1.7 CHANGE SYSTEM NAME SUBSYSTEM

The Change System Name subsystem allows the user to change from one MEDL-R data base to another without exiting the MEDL-R system. This subsystem is provided with good syntax checking of the system name requested. Changing from one data base to another is the only function performed.

This important subsystem could be expanded to perform additional functions. For instance, there is currently no provision within the MEDL-R system for combining two or more data bases into one. Such a capability would be useful in the analysis of large systems that can be conceptualized as a set of subsystems. The subsystems could be analyzed for internal consistency before combination. The capability for analysis by subsystem before integration would result in the following:

- A significant increase in run-time efficiency while Query, Analyzer, or Metric subsystem operations are

ORIGINAL PAGE IS
OF POOR QUALITY

performed--Some of the options under these subsystems involve the analysis of requirement-to-requirement relationships. The number of such relationships is proportional to the square of the total number of requirements.

- A possible reduction in errors caused by confusion--There is a limit to the number of items an individual can keep track of simultaneously. The MEDL-R system provides many aids in tracking requirements; however, these aids only raise the limit and do not eliminate it.
- A method of measuring the coupling between subsystems (possibly by use of To Be Supplied (TBS) stubs)

Another potential, but probably less useful, capability would be that of creating of a separate data base from a subset of requirements in an existing MEDL-R data base. This feature would be useful when a new system is created using a portion of a previously defined system. The capability for isolating a portion for a MEDL-R data base would result in the following:

- Cost savings derived from reuse of a set of requirements to define the baseline for a similar system
- Flexibility in the reorganization of a system into a new set of subsystems--Once a system is defined, a trial organization into subsystems would be useful in locating an organization with minimal coupling.
- A provision for a converse of the merge capability previously suggested--The user would be free to combine merging and isolating to perform reorganizations to suit his/her own particular needs.

ORIGINAL PAGE IS
OF POOR QUALITY

2.2 FLOW DIAGRAMS OF USER/SYSTEM INTERACTION

This section provides six flow diagrams (Figures 2-1 through 2-6) depicting the interaction between the user and the MEDL-R system. When compared, these figures show the relative level of user interaction for each subsystem.

In these figures, squares represent processes, ellipses contain MEDL-R prompts, and items enclosed in quotation marks represent user input. Items not in quotation marks, ellipses, or squares represent classes of responses for which knowledge of the proper syntax is required (e.g., the "valid entry" response, for which the user must know what constitutes a valid entry).

The Language Translator subsystem, representing one extreme, shows a minimum of interaction. The Update subsystem, representing the other extreme, is by far the most complex and shows a high degree of interaction. The Metric subsystem is not diagrammed because of the insufficient experience with this subsystem mentioned in Section 2.1.6.

Two conclusions can be drawn from the figures:

- The two language editor subsystems, Create and Update, differ in the level of complexity in user interaction.
- The prompting, in most cases, appears to be clear. The prompt indicates both the desired type of input and the position of the user in the system hierarchy.

2.3 STRUCTURAL ANALYSIS

This section describes and evaluates the MEDL-R system structure and data set structure.

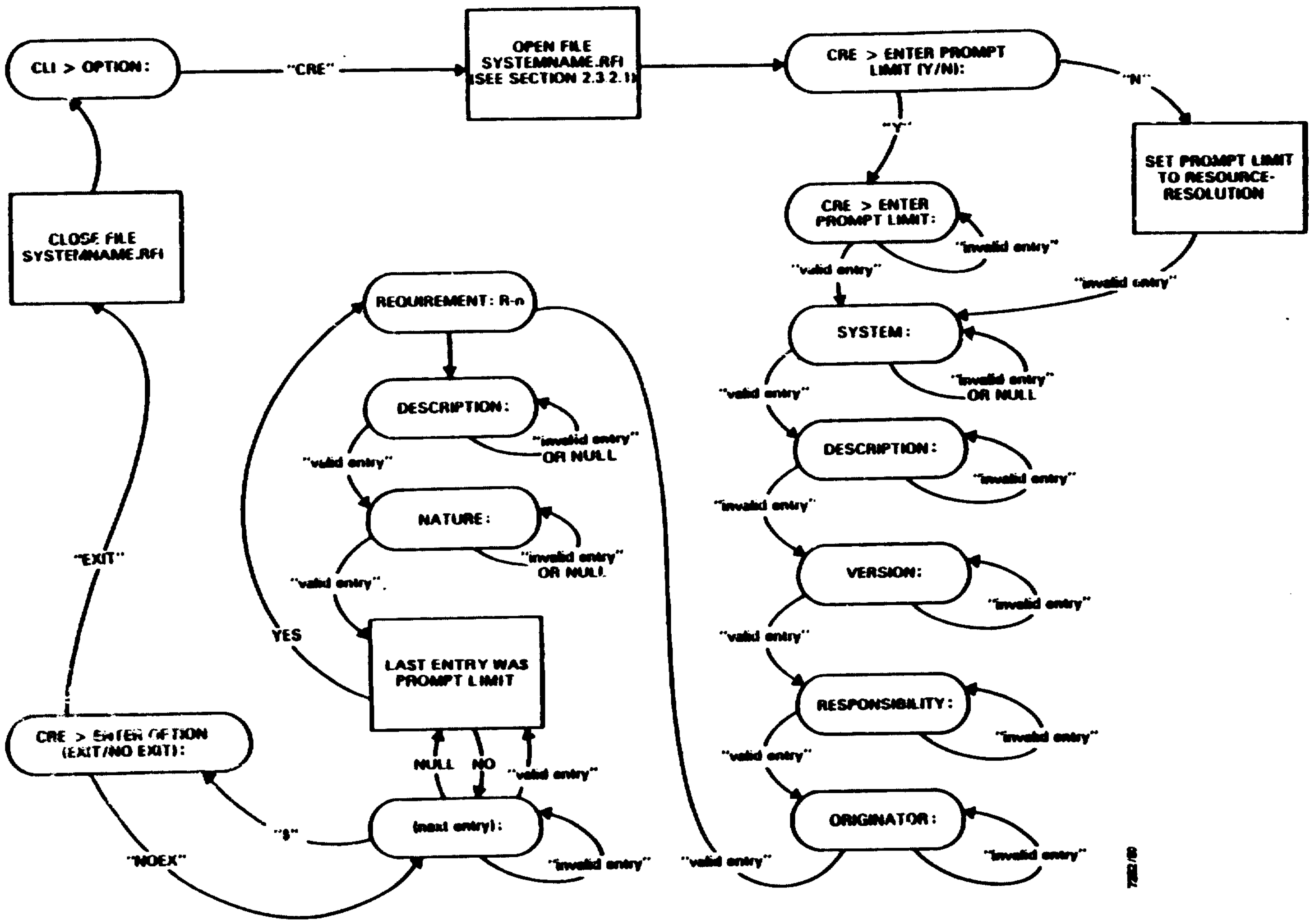

Figure 2-1. Flow of Control in Create Subsystem

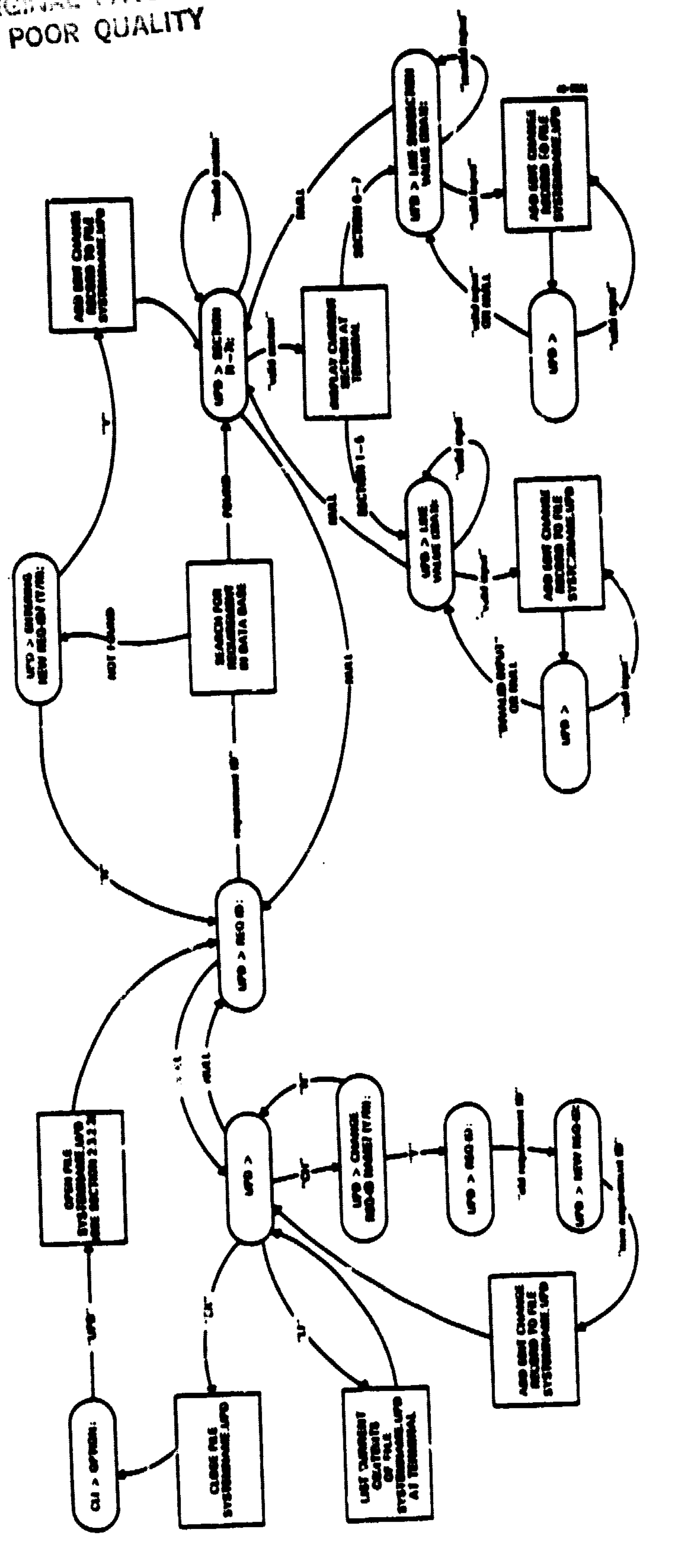
UPD >
UPD > REQ ID:
CLI > OPTION:
CLOSE FILE SYSTEMNAME.UPD
SEARCH FOR REQUIREMENT IN DATA BASE
DISPLAY CHANGE SECTION AT TERMINAL
ADD SIDE CHANGE RECORD TO FILE SYSTEMNAME.UPD
NOT FOUND
FOUND
NULL

Figure 2-2. Flow of Control in Update Subsystem

ORIGINAL PAGE IS
OF POOR QUALITY

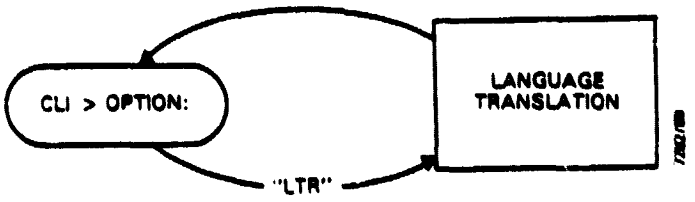

Figure 2-3. Flow of Control in Language Translator Subsystem

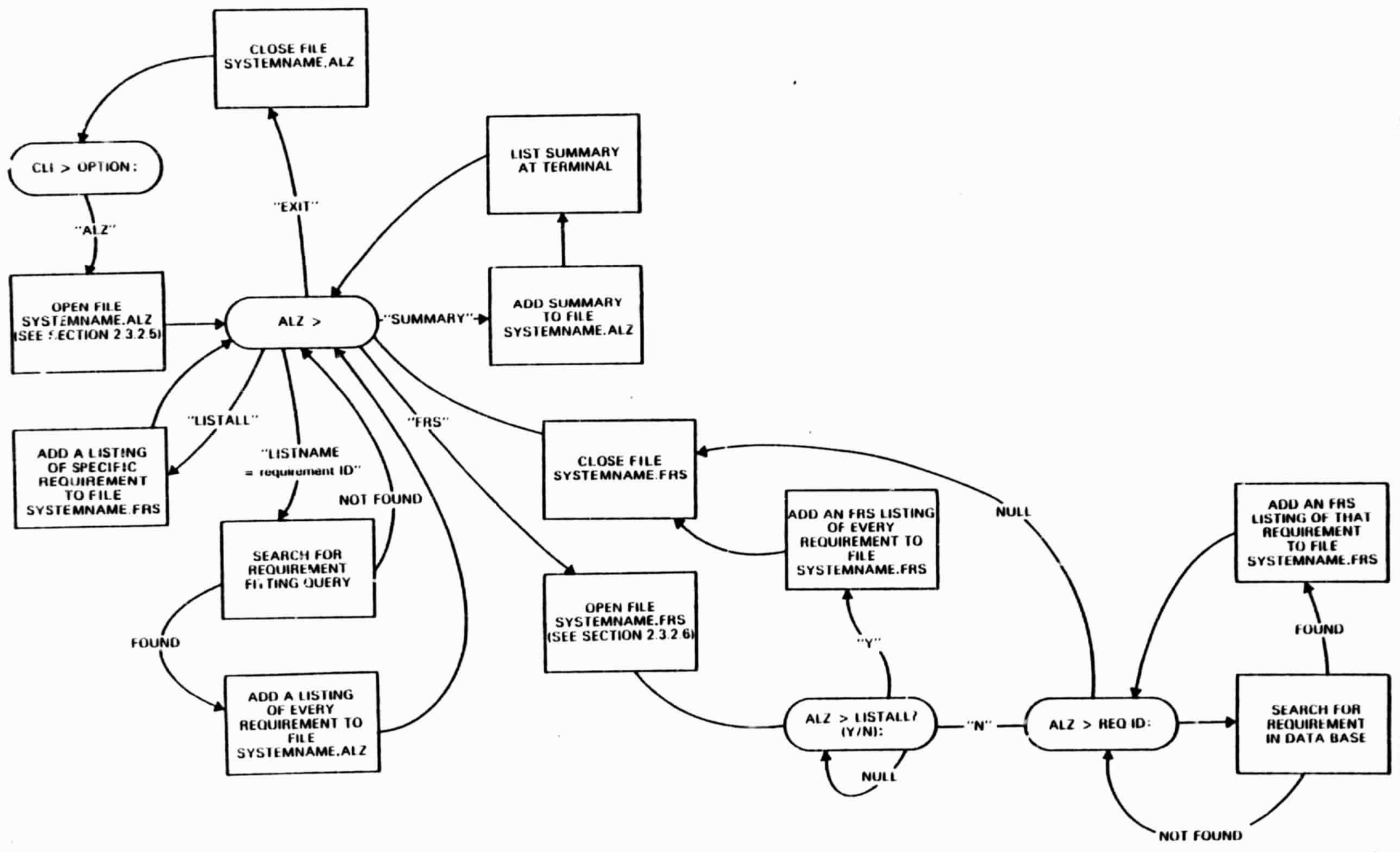
CLOSE FILE SYSTEMNAME.ALZ
CLI > OPTION:
"ALZ"
OPEN FILE SYSTEMNAME.ALZ (SEE SECTION 2.3.2.5)
"EXIT"
LIST SUMMARY AT TERMINAL
ALZ >
"SUMMARY"
ADD SUMMARY TO FILE SYSTEMNAME.ALZ
"LISTALL"
ADD A LISTING OF SPECIFIC REQUIREMENT TO FILE SYSTEMNAME.FRS
"LISTNAME = requirement ID"
"FRS"
NOT FOUND
SEARCH FOR REQUIREMENT FITTING QUERY
FOUND
ADD A LISTING OF EVERY REQUIREMENT TO FILE SYSTEMNAME.ALZ
CLOSE FILE SYSTEMNAME.FRS
OPEN FILE SYSTEMNAME.FRS (SEE SECTION 2.3.2.6)
ADD AN FRS LISTING OF EVERY REQUIREMENT TO FILE SYSTEMNAME.FRS
"Y"
ALZ > LISTALL? (Y/N):
NULL
"N"
NULL
ALZ > REQ ID:
ADD AN FRS LISTING OF THAT REQUIREMENT TO FILE SYSTEMNAME.FRS
FOUND
SEARCH FOR REQUIREMENT IN DATA BASE
NOT FOUND

Figure 2-4. Flow of Control in Analyzer Subsystem

ORIGINAL PAGE IS
OF POOR QUALITY

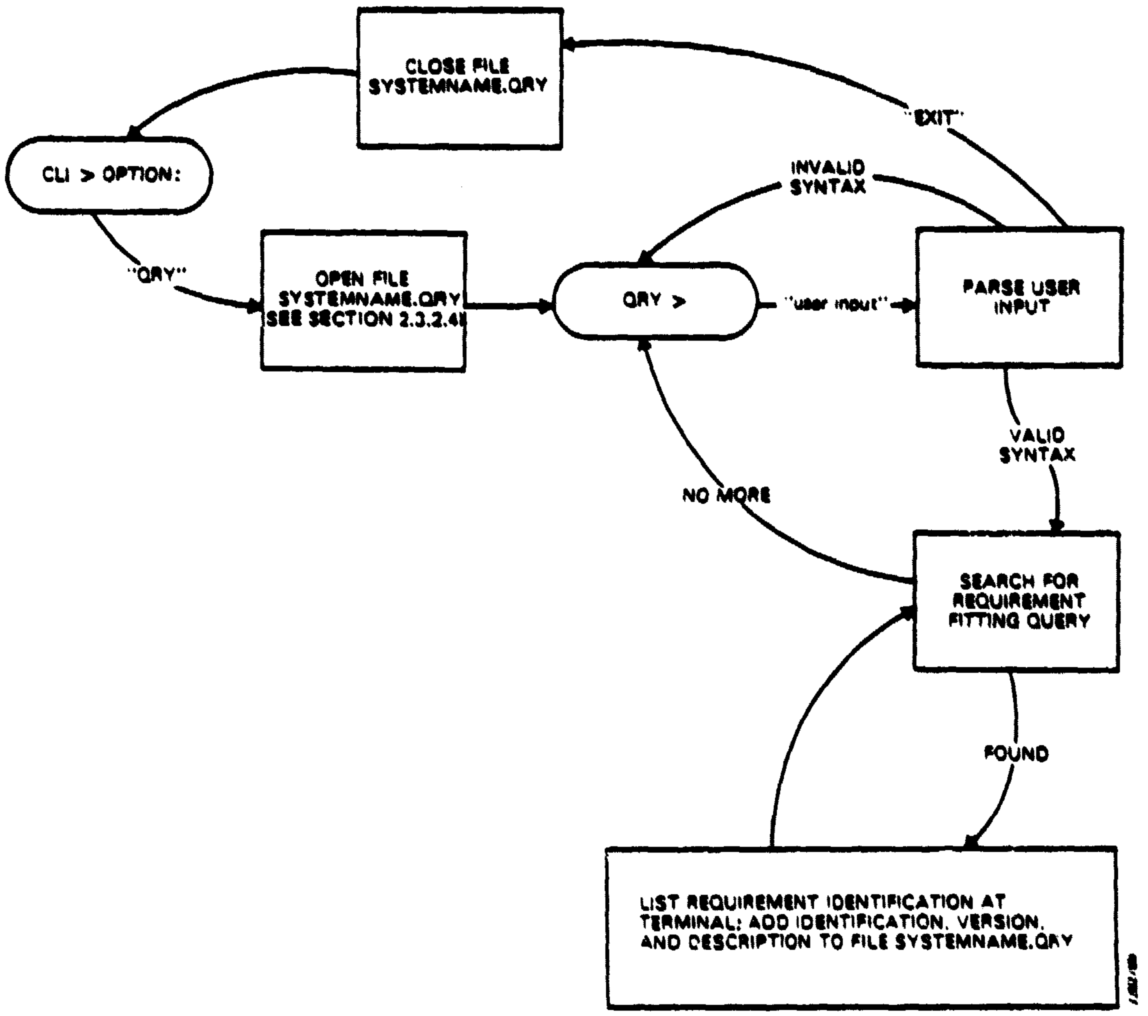

Figure 2-5. Flow of Control in Query Subsystem

ORIGINAL PAGE IS
OF POOR QUALITY

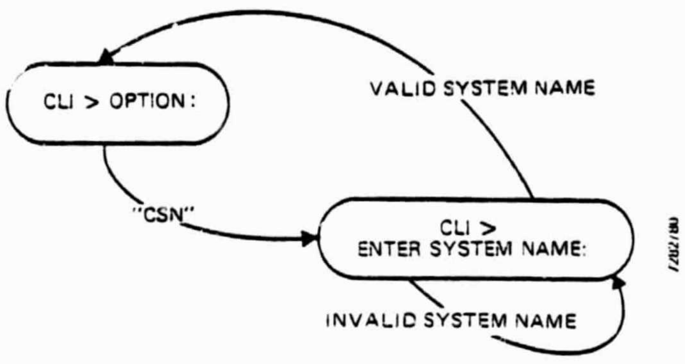

Figure 2-6. Flow of Control in Change System Name Subsystem

ORIGINAL PAGE IS
OF POOR QUALITY

2.3.1 MEDL-R SYSTEM STRUCTURE

The source code for the MEDL-R system is written entirely in the PDP-11 FORTRAN IV PLUS language. The system is highly modular, and the functional components are easily matched to subsystem module hierarchies.

Each module is coded using system-wide naming conventions; this is aided by the liberal use of the INCLUDE compiler directive for COMMON block code. A marginally acceptable technique of using the INCLUDE directive for repetitive executable code (notably error recovery sequences) is also used.

The source code is complete even to the extent of FORTRAN routines from the Martin Marietta Aerospace Storage System (MASS).

2.3.2 MEDL-R DATA SET STRUCTURE

This section describes the function, content, and format of all MEDL-R system data files and the specific accesses to each file by the various MEDL-R subsystems.

The file-naming convention used by the MEDL-R system is as follows:

    Systemname.XXX

where Systemname is the user's name for the collection of requirements to be analyzed by the MEDL-R system (this name is supplied to the Command Language Interpreter control module and is used to create the PDP-11 file structure), and XXX is the data file type. The data files are described in Section 2.3.2.1 through 2.3.2.8.

2.3.2.1 RFI File

The RFI file contains the initial requirements for a new MEDL-R system. The file format is sequential fixed-length card image records, each containing ASCII data only. The disk name of this file is Systemname.RFI.

The RFI file is created or extended by the Create subsystem.[1] The presence of this file is determined by the Command Language Interpreter control module, and the presence or absence of the RFI file is used to allow or disallow some user options (see Reference 1, page II-2). The RFI file is read by the Language Translator subsystem and is deleted upon the successful completion of a data base build by that subsystem.

The RFI file is useful if a set of requirements is created offline. The usefulness would be even greater if an option existed to create an RFI-like file from a subset of requirements in an existing MEDL-R data base.

2.3.2.2 UPD File

The UPD file contains the updates to an existing MEDL-R system. The file format is sequential fixed-length card image records, each containing ASCII data only. The disk name of this file is Systemname.UPD.

The UPD file is created or extended by the Update subsystem. The presence of this file is determined by the Command Language Interpreter control module, and the presence or absence of the UPD file is used to allow or disallow some user options (see Reference 1, page II-2). The UPD file is read by the Language Translator subsystem and is deleted upon the successful completion of a data base update by that subsystem.

2.3.2.3 LTR File

The LTR file is the listing of the Language Translator subsystem actions for each data base build or update. This

[1] If the FORMAT of the input card image were known, the file could be created offline.

ORIGINAL PAGE IS
OF POOR QUALITY

file contains a copy of the data input to the Language Translator subsystem and any warning and error messages generated in response to the input. The listing is terminated with a summary of errors, statistics, and a completion message. The file format is sequential variable-length records with carriage control. The disk name of this file is Systemname.LTR.

A new version of this file is created each time the user invokes the Language Translator subsystem. After the MEDL-R session, the user may list the LTR file version(s) at the terminal or on the lineprinter.

2.3.2.4 QRY File

The QRY file is the output listing of the Query subsystem. The file contains a copy of the user's QUERY request followed by the IDENTIFICATION, DESCRIPTION, and VERSION of each requirement that matches the QUERY (see Sections 3.1.1, 3.1.2, and 3.1.7 for the definitions of IDENTIFICATION, DESCRIPTION, and VERSION, respectively). The file format is sequential variable-length records with carriage control. The disk name of this file is Systemname.QRY.

A new version of this file is created each time the user invokes the Query subsystem. After the MEDL-R session, the user may list the QRY file version(s) at the terminal or on the lineprinter.

2.3.2.5 ALZ File

The ALZ file is one of two output listings of the Analyzer subsystem. The file contains the reports generated by the SUMMARY, LISTALL, and LISTNAME commands (see Section 2.1.5). The file format is sequential variable-length records with carriage control. The disk name of this file is Systemname.ALZ.

A new version of this file is created each time the user invokes the Analyzer subsystem and uses a SUMMARY, LISTALL, or LISTNAME command. After the MEDL-R session, the user may list the ALZ file version(s) at the terminal or on the lineprinter.

2.3.2.6 FRS File

The FRS file is one of the two output listings of the Analyzer subsystem. The file contains the report generated by the FRS command (see Section 2.1.5). The file format is sequential variable-length records with carriage control. The disk name of this file is Systemname.FRS.

A new version of this file is created each time the user invokes the Analyzer subsystem and uses the FRS command. After the MEDL-R session, the user may list the FRS file version(s) at the terminal or on the lineprinter.

2.3.2.7 MAT, ADJ, WGT, DIS, and SEQ Files

The MAT, ADJ, WGT, DIS, and SEQ files are storage files for partial or intermediate forms of the matrices used in the Metric subsystem. The format of each file is sequential variable-length binary records. The disk names of these files are Systemname.MAT, Systemname.ADJ, Systemname.WGT, Systemname.DIS, and Systemname.SEQ.

The creation and reuse of these files are controlled by the STORE and RELOAD commands within the Metric subsystem. As mentioned in Section 2.1.6, little experience has been gained with this subsystem.

2.3.2.8 AL1, AL2, AL3, RE1, RE2, and RE3 Files

The AL1, AL2, AL3, RE1, RE2, and RE3 files form the central MEDL-R data base. These files contain all information supplied by the user. The formats are direct-access fixed-length binary records. The disk names of these files are

ORIGINAL PAGE IS
OF POOR QUALITY

Systemname.AL1, Systemname.AL2, Systemname.AL3, Systemname.RE1, Systemname.RE2, and Systemname.RE3.

The data base is created or modified by the Language Translator subsystem through the use of MASS utility routines. The presence of the data base is determined by the Command Language Interpreter control module, and its presence or absence is used to allow or disallow some user options (see Reference 1, page II-2). The data base is read by the Query, Analyzer, and Metric subsystems as the source of information for all processing performed by these subsystems.

The MEDL-R data base is organized into two groups of three files each. The first group, called relation ALLNAMES, consists of the AL1, AL2, and AL3 files. The second group, called relation RELS, consists of the RE1, RE2, and RE3 files.

The first file of each group (AL1 or RE1) is termed a Tuple Description Table (TDT). This file contains pointers to the end of the other two files in the group. The TDT also points to the Alphanumeric Data File (discussed below), a section of which describes the organization of data in the files.

The second file of each group (AL2 or RE2) is termed a Tuple File (TF). This file contains links that describe the relationship between data items. The AL2 file links a sequential key field through a MEDL-R requirement entry-type code to a pointer to an entry in the alphanumeric data. The RE2 file links every record (entered by the user) from the requirement IDENTIFICATION through a MEDL-R requirement entry-type code to a pointer to the first occurrence of the actual alphanumeric data.

The third file of each group (AL3 or RE3) is termed an Alphanumeric Data File (ADF). This file contains two sections. The first section contains alphanumeric tags that describe the fields of a record in the associated TF. This section

ORIGINAL PAGE IS
OF POOR QUALITY

is overhead and is simply an extension of the TDT that points to it. (The presence of this section is probably a consequence of the restriction of alphanumeric data to the ADF.) This section is present in both the AL3 and RE3 files.

The second section (present in AL3 only) contains an example of each MEDL-R keyword, string, or text entered by the user.

A simplified schematic of the organization of the MEDL-R data base is shown in Figure 2-7. The data base is shown after the MEDL-R source has been passed from the Create subsystem to the Language Translator subsystem. (The sample of the MEDL-R source is not complete, since the Create subsystem demands that the DESCRIPTION and NATURE entries always be present; the source shown is intended as a simple example.) The user has specified the STATUS of both requirement R-1 and R-2 as ACTIVE and the STATUS of R-3 as SOFT. The linking pointers start in RE2 and can be traced through AL2 to AL3.

The same data base is shown after an update in Figure 2-8. The user has employed the Update subsystem to change the STATUS of requirement R-2 to OBSOLETE and then invoked the Language Translator subsystem. (For clarity, the sample of MEDL-R source is not shown in the UPD file format.)

The organization of MASS files is quite flexible and should be adaptable to any of the proposed modifications to the MEDL-R system.

The source code for the standalone utility SUMREL is supplied with the MEDL-R system. SUMREL produces a formatted dump of either the AL1, AL2, and AL3 files or the RE1, RE2, and RE3 files.

Because MASS is only mentioned, and not cited, in the MEDL-R documentation, SUMREL listings were used to analyze the structure of the MEDL-R data base.

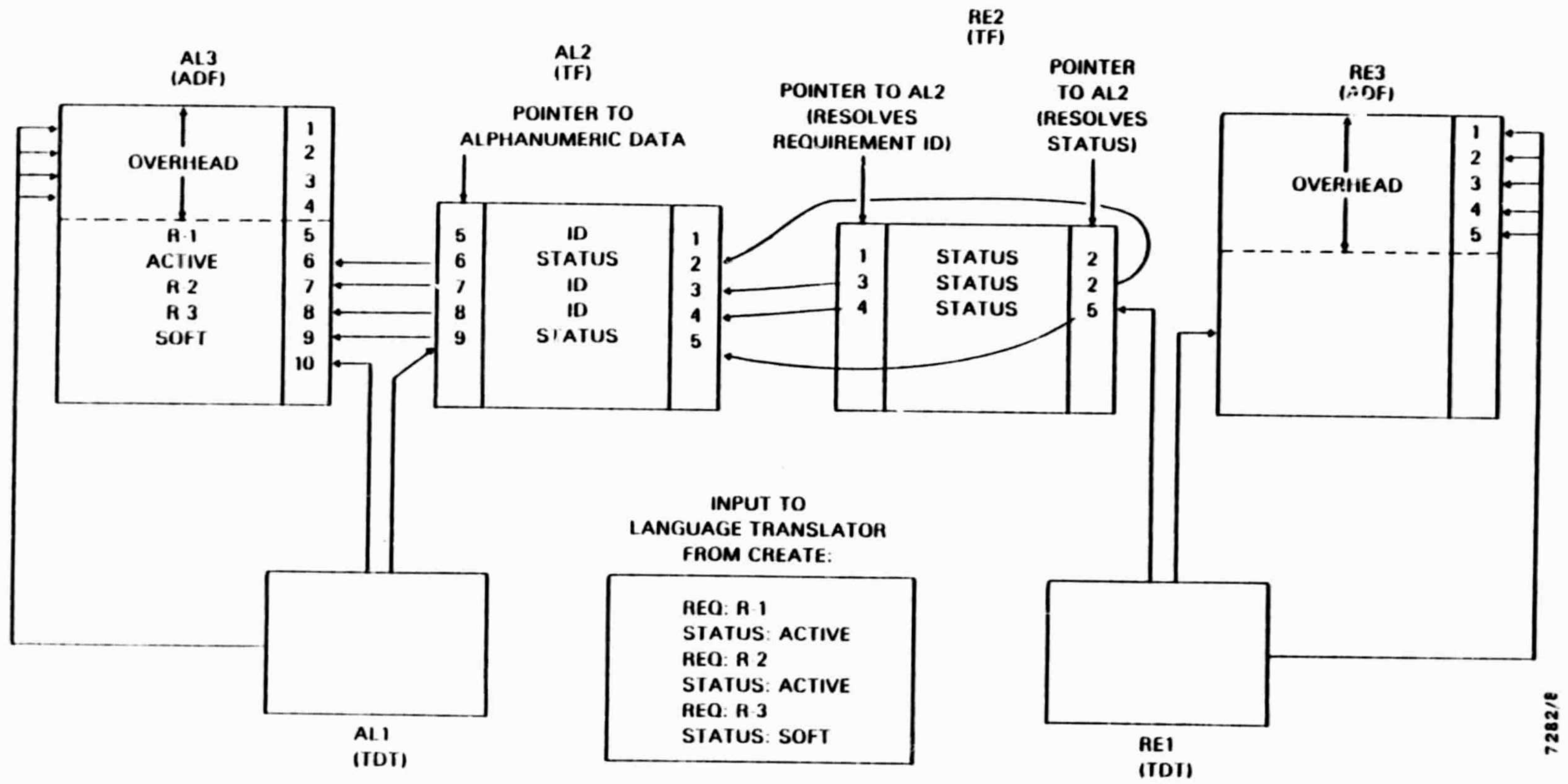

Figure 2-7. Schematic Representation of MEDL-R Data Base After Create Subsystem Action

ORIGINAL PAGE IS
OF POOR QUALITY

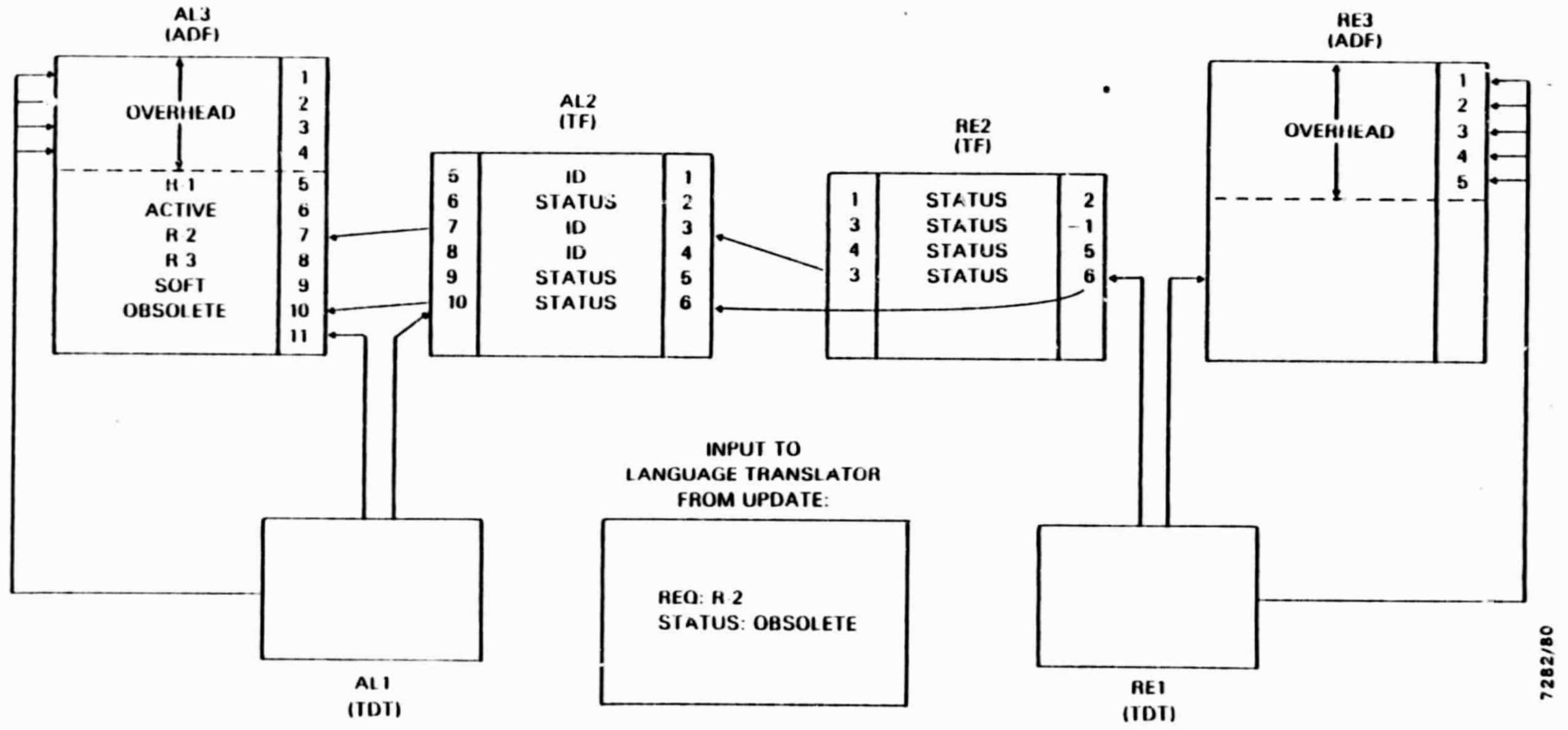

Figure 2-8. Schematic Representation of MEDL-R Data Base After Update Subsystem Action

ORIGINAL PAGE IS
OF POOR QUALITY

2.4 RECOMMENDED MEDL-R STRUCTURE

This section specifies a recommended structure and data flow for the MEDL-R system based on the comments in Sections 2.1 through 2.3. This recommended structure should allow the system to accept new language features and new analysis capabilities.

As previously stated, the current structure of centralizing the operations around the language translation step is an excellent scheme. The function division of the modules into Create, Update, Query, Analyzer, and Metric subsystems is also good. The use of the MASS data base structure is equally good (although better documentation is needed if serious work in modifying it is to be accomplished). The basic criticisms of the structure, specified below, are details of design and implementation:

- There should be only one editor (used for both new input and modifications to old).
- Upon entering or modifying a requirement, the language translation should take place immediately without user command. This eliminates many files, reduces complexity, and gives the user immediate feedback as to the validity of the entered requirements.
- The only saved files should be the requirements data base. All other files should be directed to the user terminal or lineprinter and should be deleted after used.
- The Analyzer subsystem should be renamed the Report Generator subsystem, and the Metric subsystem is really the first analysis module.

Figure 2-9 shows a structure that meets the requirements specified above. The only new process identified is that of

ORIGINAL PAGE IS
OF POOR QUALITY

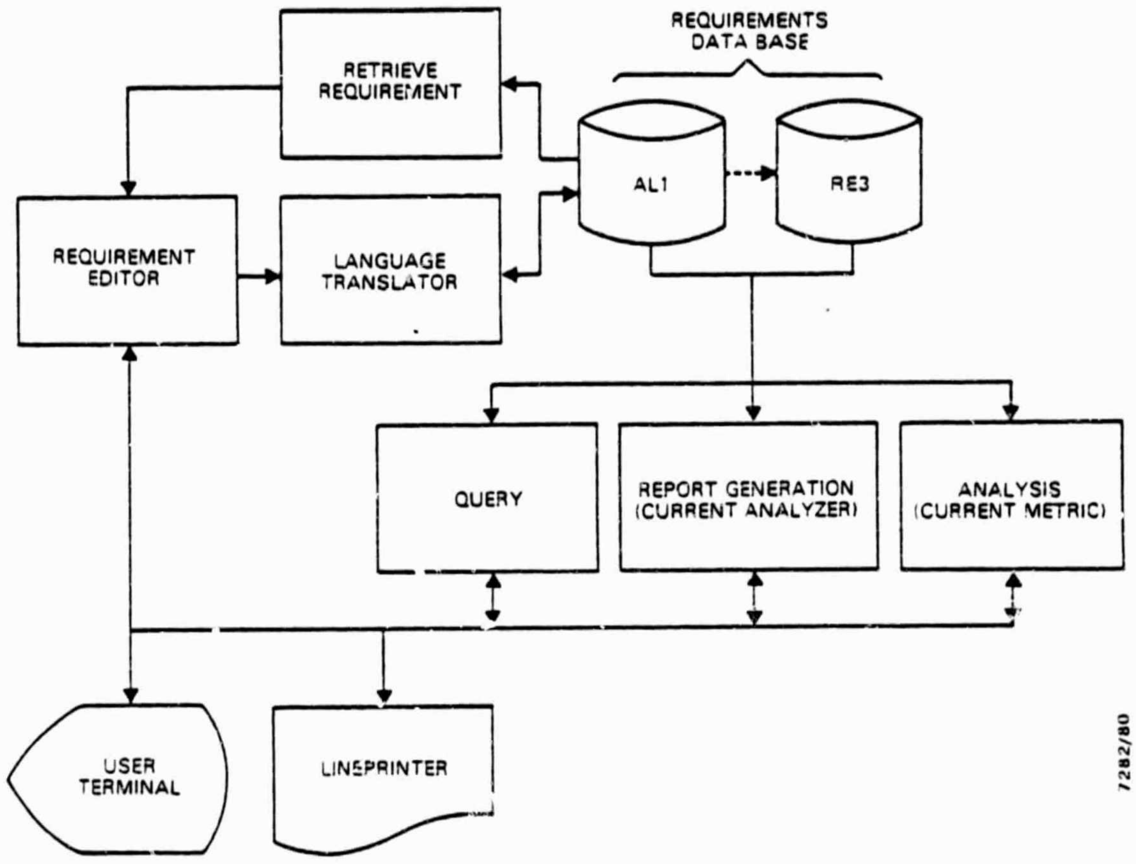

Figure 2-9. Recommended MEDL-R System Structure

retrieving a requirement from the requirements data base. This allows the editor to modify the requirement (if it currently exists).

SECTION 3 - MEDL-R LANGUAGE SYNTAX

This section summarizes the language syntax currently used by the MEDL-R system. Section 3.1 evaluates the language elements as to their applicability, completeness, structure, strengths, and weaknesses. Section 3.2 specifies recommended enhancements to the language.

3.1 CURRENT MEDL-R LANGUAGE SYNTAX

The current version of the MEDL-R system expresses system requirements according to the following structure:

- A system is a data base that contains the specifications (requirements) for one potential problem definition.
- A requirement is an individual data structure within the data base. A requirement is composed of up to 21 types of information (entries) that define, clarify, or categorize that requirement.
- An entry is the smallest unit of data that is accessible to the MEDL-R system. Each of the 21 possible entries must be of the form (type) specified for that particular entry.
- The type of an entry may be text, keyword, or string. Table 3-1 specifies the rules for correctly forming each type and the entries allowed for each type.

Table 3-2 summarizes the 21 requirement entries supported by the current version of the MEDL-R system. Table 3-3 lists the keywords currently available. The entries and the keywords are specifically defined in Reference 2.

The use of three entry types makes the MEDL-R system syntax flexible. The text type permits the storage of information that is readable, understandable, and meaningful both to

Table 3-1. Current Requirement Type Syntax

TYPE	DESCRIPTION
TEXT	ANY LINE OF TEXT IS ACCEPTED. TEXT IS TERMINATED BY TWO CONSECUTIVE PERIODS. DESCRIPTION AND EXPLANATION ARE THE ONLY TEXT ENTRIES.
KEYWORD	KEYWORDS ARE PREDEFINED TERMS USED TO CLASSIFY THE REQUIREMENT. (THE SPECIFIC KEYWORDS ARE LISTED IN TABLE 3-3.) THE RULES FOR SPECIFYING A KEYWORD ARE AS FOLLOWS. THE EXACT SPELLING (AS SHOWN IN TABLE 3-3) (UP TO 16 CHARACTERS) IS REQUIRED FOR EACH KEYWORD. NO EMBEDDED BLANKS, COMMAS, OR PERIODS ARE ALLOWED. COMMAS SEPARATE KEYWORDS APPEARING IN LISTS. PERIODS TERMINATE LISTS OR SINGLE KEYWORDS. THE NATURE AND RESULTING-FROM ENTRIES ALLOW LISTS OF KEYWORDS. THE STATUS AND SCOPE ENTRIES ALLOW THE SELECTION OF ONLY ONE KEYWORD.
STRING	A STRING IS A USER-DEFINED SEQUENCE OF UP TO 30 CHARACTERS. THE FIRST CHARACTER MUST BE ALPHABETIC. THE RULES FOR VALID CHARACTERS, SEPARATORS, AND TERMINATORS ARE THE SAME AS FOR KEYWORDS. THE IDENTIFICATION, RESPONSIBILITY, ORIGINATOR, VERSION, SUBSYSTEM, AND SOURCE ENTRIES ALLOW ONLY ONE STRING. THE SUBJECT, REPLACES, REPLACED-BY, DERIVES, DERIVED-FROM, FUNCTION-RESOLUTION, DATA-RESOLUTION, AND RESOURCE-RESOLUTION ENTRIES ALLOW LISTS OF STRINGS. THE CONSTRAINT ENTRY REQUIRES A SPECIAL STRING, WHICH IS DESCRIBED ON PAGE III-13 OF REFERENCE 2. THIS ENTRY IS REQUIRED ONLY WHEN THE NATURE KEYWORD PERFORMANCE IS SPECIFIED.

7282/80

ORIGINAL PAGE IS
OF POOR QUALITY

Table 3-2. Current Requirement Entries

ENTRY	TYPE	LIST[1]	MEANING
IDENTIFICATION	STRING		NAME TAG OF THIS REQUIREMENT
DESCRIPTION	TEXT		ENGLISH EXPRESSION OF REQUIREMENT
NATURE	KEYWORD	L	CATEGORY
RESPONSIBILITY	STRING		NAME OF PERSON, GROUP
ORIGINATOR	STRING		NAME OF PERSON, GROUP
SCOPE	KEYWORD		RANGE OF INFLUENCE
VERSION	STRING		AUDIT
SUBSYSTEM	STRING		PART OF SYSTEM
SOURCE	STRING		DOCUMENT REFERENCE
CONSTRAINT	STRING		QUANTITY
RESULTING-FROM	KEYWORD	L	MOTIVATION
SUBJECT	STRING	L	USER CATEGORY
EXPLANATION	TEXT		MISCELLANEOUS INFORMATION
STATUS	KEYWORD		CURRENT STANDING
REPLACES	STRING	L	OVERRIDDEN REQUIREMENT
REPLACED-BY	STRING	L	OVERRIDING REQUIREMENT
DERIVES	STRING	L	SUBSEQUENT REQUIREMENT
DERIVED-FROM	STRING	L	ORIGINATING REQUIREMENT
FUNCTION-RESOLUTION	STRING	L	MODULE NAME
DATA-RESOLUTION	STRING	L	DATA SET NAME
RESOURCE-RESOLUTION	STRING	L	SPECIFIC HARDWARE

7282/80

[1] AN L IN THIS COLUMN INDICATES THAT ONE OR MORE STRINGS OR KEYWORDS MAY BE PRESENT IN THE ASSOCIATED ENTRY.

Table 3-3. Current Keywords

ENTRY	KEYWORDS
NATURE	DEVELOPMENT TECHNIQUES TOOLS METHODOLOGY MANAGEMENT MANPOWER BUDGET SCHEDULES END-ITEMS FACILITY HARDWARE OPERATING-SYSTEM LANGUAGES PRODUCT INTERNAL PROCEDURAL STRUCTURAL TEMPORAL DATA INTERFACE EXTERNAL OPERATIONAL PERFORMANCE USER-INTERFACE TGT-FACILITY HARDWARE OPERATING-SYSTEM LANGUAGES EXISTING-SOFTWARE
SCOPE	GLOBAL LIMITED
RESULTING-FROM	COMPANY-GOALS COMPANY-STANDARDS CUSTOMER-DIRECTION CUSTOMER-STANDARDS REAL-WORLD-MODEL QUALITY-CONSIDERATIONS ECONOMICS POLITICS
STATUS	ACTIVE SOFT OBSOLETE

7282/80

ORIGINAL PAGE IS
OF POOR QUALITY

the developer and to the originator of a software system. The keyword type allows the MEDL-R system to impose a built-in set of categories upon requirements. The string type supplies a common format for requirement names (which can be used for tags and pointers) and for user-defined quantities (names of people, specialized categories, or actual items). No suggested improvement to the MEDL-R system involves an enlargement of the set of permissible types.

Sections 3.1.1 through 3.1.21 describe the MEDL-R system entries. (Each entry is described in detail in Reference 2.) Each section includes (where applicable) a specific analysis of the entry's applicability, completeness, structure, strengths, and weaknesses. Many of these factors are considered in measuring the usability of the entry in current or future analysis. Appropriate recommendations concerning how and when to use the individual language elements are also provided.

3.1.1 IDENTIFICATION

The IDENTIFICATION entry is the means of referring to a particular requirement from another requirement or by any of the existing analysis procedures.

This entry may be supplied by the MEDL-R system (in the Create mode) or by the user (in the Update mode). When supplied by the MEDL-R system, the IDENTIFICATION sequence is R-1, R-2, and so forth. Duplicate IDENTIFICATIONs are generated by the MEDL-R system if the Create subsystem is used more than once before processing proceeds to the Language Translator subsystem. In the Update mode, any valid string may be used for IDENTIFICATION. This is the preferred naming method, because the user may label the requirement with a name indicative of its contents.

3.1.2 DESCRIPTION

The DESCRIPTION entry contains the English-language expression of the requirement. The entire requirement is dependent upon a clear and unambiguous statement in this entry. DESCRIPTION is not currently subjected to analysis by the MEDL-R system (except for text-fragment searches by the Query subsystem). DESCRIPTION is expanded, annotated, qualified, categorized, and so forth, by the remaining entries of the requirement. DESCRIPTION is the core of the MEDL-R system concept. The user, however, may benefit from the following suggestions. The text entered here should be both concise and clear. Conciseness is desirable because the QRY data set will contain a complete copy of the DESCRIPTION entry of every matching requirement each time it matches a query. Clarity is desirable because these DESCRIPTIONs will be reported alone and out of context. DESCRIPTION is, as it should be, a mandatory entry to form a requirement.

3.1.3 NATURE

The NATURE keyword is the primary classification of the requirement. The current set of available keywords (see Table 2-3) attempts to present a universal classification scheme; that is, every possible requirement should fit under one or more of the keywords.

The NATURE keyword set is one of the most important features of the MEDL-R system. A user should fully understand the meanings of each of the keywords and should state the requirements with the keywords in mind. In this way, the keywords can indicate which areas additional requirements should address. This generality leads to a criticism of the current NATURE keyword set. A user should be able to tailor the set to match inherent qualities of a specific type of system. The current set is too general and provides limited guidance for a specific application.

A second criticism of the NATURE entry is that although the entry contains an implicit structure, this structure is not implemented or enforced in the MEDL-R system. Figure 3-1 shows the implied structure as a hierarchy. Currently, a user may tag a requirement with the NATURE keyword LANGUAGES. The hierarchy list shows that this is ambiguous. A requirement tagged with both DEVELOPMENT and EXISTING-SOFTWARE is obviously incorrect, but this tagging is currently allowed by the MEDL-R system. On the other hand, a requirement tagged with DEVELOPMENT, FACILITY, and HARDWARE (enforcing the hierarchy) is clearly different from a requirement tagged with PRODUCT, TGT-FACILITY, and HARDWARE.

A third criticism is that it is difficult to remember the spelling and definition of all NATURE keywords. The user must either use the list presented in the user's guide (Reference 2) as a reference or rely solely upon a remembered subset.

3.1.4 RESPONSIBILITY

The RESPONSIBILITY entry tracks the implementation of the requirement into the design phase.

RESPONSIBILITY may be specified in the Query mode, thus collecting, as a group, those requirements that will be resolved by a particular person or group.

3.1.5 ORIGINATOR

The ORIGINATOR entry tracks a requirement back to its origin. Tracking can be very important if questions arise concerning the requirement's function, motivation, content, and so forth.

ORIGINATOR is not available in the Query mode. Thus, it is difficult to collect one originator's requirements into one package (e.g., to submit a request for validation of the finalized requirements to each originator).

- DEVELOPMENT
    - TECHNIQUES
        - TOOLS
        - METHODOLOGY
    - MANAGEMENT
        - MANPOWER
        - BUDGET
        - SCHEDULES
        - END-ITEMS
    - FACILITY
        - HARDWARE
        - OPERATING-SYSTEM
        - LANGUAGES
- PRODUCT
    - INTERNAL
        - PROCEDURAL
        - STRUCTURAL
        - TEMPORAL
        - DATA
        - INTERFACE
    - EXTERNAL
        - OPERATIONAL
        - PERFORMANCE
        - USER-INTERFACE
    - TGT-FACILITY
        - HARDWARE
        - OPERATING-SYSTEM
        - LANGUAGES
        - EXISTING-SOFTWARE

7262/80

Figure 3-1. Current NATURE Keywords in Hierarchical Form

3.1.6 SCOPE

The SCOPE entry indicates how the requirement is to relate to the system. The current choice of GLOBAL or LIMITED keywords for SCOPE tends to be meaningless unless the boundaries of influence are in some way indicated when LIMITED is specified. The SCOPE entry is thus not fully used by the MEDL-R system language, and the explanation of this entry in Reference 2 does not help the user to understand its use.

3.1.7 VERSION

The VERSION entry is intended to help determine the stage of the system at which the requirement was added.

Because this entry is supplied by the user, misspellings, inaccuracies, or blunders will negate its purpose. Currently, the MEDL-R system does not assist in this housekeeping chore.

3.1.8 SUBSYSTEM

The SUBSYSTEM entry supplies a name for the particular portion of the system to which the requirement applies.

This entry is tied strongly to the SCOPE entry concept. The MEDL-R system should recognize this connection and prompt the user to supply valid relationships.

3.1.9 SOURCE

The SOURCE entry traces the requirement to a written document and is thus very important.

If the MEDL-R system is used to enter and analyze a set of requirements (and not as a tool to help develop the requirements), this entry can be used to determine whether the requirement was entered correctly. In developing requirements, SOURCE can be used to refer to supporting evidences.

ORIGINAL PAGE IS
OF POOR QUALITY

3.1.10 CONSTRAINT

The CONSTRAINT entry allows for a "quantitative restriction" as a requirement. It is permitted only when NATURE keyword PERFORMANCE is supplied. The quantity must be expressed as a rate (e.g., 30-FRAMES/SECOND). Currently, there is no analysis of this entry. This entry was apparently planned for some purpose other than further clarification of the requirement (as seen by the restriction on entry units). The present restriction to express the quantity as a rate results in confusion when this cannot be done (e.g., if the user wishes to note a requirement for the amount of in-core memory).

3.1.11 RESULTING-FROM

The RESULTING-FROM entry indicates the motivation for including the requirement. The current set of available keywords is specified in Table 3-3.

The name of this entry may cause it to be confused with the DERIVED-FROM entry.

RESULTING-FROM keywords are hard to remember and do not cover all reasons for including a requirement. As with the NATURE keywords, this set of keywords should be tailored to the specific motivations important to the user.

3.1.12 SUBJECT

The SUBJECT entry is intended for user-supplied keywords. Any valid string or strings may be entered. Such an entry gives the user a chance to cross-reference requirements in terms specific to the system.

The SUBJECT entry is a potentially useful device. The user should select a limited number of SUBJECT strings and try to stick to them. This procedure should limit the problems with input errors and misspellings. Such input errors and

ORIGINAL PAGE IS
OF POOR QUALITY

misspellings can be serious because the MEDL-R system does not validate this entry.

The SUMMARY report produced by the Analyzer subsystem contains a list of SUBJECT strings and should be checked frequently to detect the presence of misspellings.

3.1.13 EXPLANATION

The EXPLANATION entry contains any expanded text relating to the requirement. The EXPLANATION entry is used to supplement the DESCRIPTION entry.

The EXPLANATION entry is not reported in Query mode, and thus the restriction of conciseness suggested in Section 3.1.2 does not apply. Any applicable text should be entered into EXPLANATION, including comments, questions, historical data, and so forth. EXPLANATION is not subjected to Query mode analysis.

3.1.14 STATUS

The STATUS entry indicates the current standing of the requirement as it is applicable to the current system.

The STATUS keywords are specified in Table 3-3. The keyword SOFT is not an accepted term and might not have a distinct meaning to all users. The keyword ACTIVE is also unclear due to the many different shades of meaning associated with the word. The STATUS keywords should indicate the progress of a requirement from proposal to final acceptance.

The keyword OBSOLETE does not, as it should, automatically remove a requirement from inclusion by all analysis subsystems.

3.1.15 REPLACES

The REPLACES entry indicates that the current requirement overrides another requirement. The string or strings entered declare the overridden requirement's IDENTIFICATION.

ORIGINAL PAGE IS
OF POOR QUALITY

Currently, this entry is not used by the MEDL-R system to its full extent. The user is responsible for making the corresponding REPLACED-BY entry in the overridden requirements. In addition, the user probably should change the overridden requirements' STATUS entries to OBSOLETE. These actions are very important in maintaining a consistent MEDL-R data base.

3.1.16 REPLACED-BY

The REPLACED-BY entry indicates that the current requirement has been overridden by another requirement. The string or strings entered declare the overriding requirement's IDENTIFICATION.

This entry is the complement of the REPLACES entry; the comments concerning that entry also apply here.

An implied relationship exists between this entry and the STATUS entry. A replaced requirement should automatically be given a STATUS of OBSOLETE.

3.1.17 DERIVES

The DERIVES entry indicates another requirement in the system that has evolved from the current requirement. The string or strings entered declare the evolved requirement's IDENTIFICATIONs.

The DERIVES entry provides the link(s) needed when a technique of decomposition is used to develop a set of system requirements.

Currently, this entry is not used by the MEDL-R system to its fullest extent. The user is responsible for making the corresponding DERIVED-FROM entry in the evolved requirement, as well as for ensuring that the evolved requirement actually exists. Again, checks such as this would help in maintaining a consistent data base, as well as flag potential errors in the requirements.

3.1.18 DERIVED-FROM

The DERIVED-FROM entry indicates that the current requirement has evolved from another requirement. The string or strings entered declare the "parent" requirement's IDENTIFICATION.

This entry is the complement of the DERIVES entry; the comments concerning that entry also apply here.

3.1.19 FUNCTION-RESOLUTION

The FUNCTION-RESOLUTION entry is a link with the design phase. The string or strings entered are names of actual modules designed to resolve the requirement.

The usefulness of this entry depends upon the implementation of the Multi-Level Expression Design Language - Design Level (MEDL-D) system. The need for this entry in the requirement statement or analysis phase is doubtful, and the entry should be avoided to maintain a clear separation between specification and design work.

3.1.20 DATA-RESOLUTION

The DATA-RESOLUTION entry is a link with the design phase. The string or strings entered are names of actual data sets that contain information specified by, or needed to implement, the requirement.

The comments concerning FUNCTION-RESOLUTION also apply to this entry.

3.1.21 RESOURCE-RESOLUTION

The RESOURCE-RESOLUTION entry is a link with the design phase. The string or strings entered are names for the designed system resources (hardware, time, or space) needed to resolve the requirement.

The comments concerning FUNCTION-RESOLUTION also apply to this entry.

3.2 RECOMMENDED ENHANCEMENTS TO MEDL-R LANGUAGE

The recommendations for enhancing the MEDL-R language fall into three categories: organization, additions, and deletions.

The recommended reorganization of requirement entries is presented in Table 3-4. The hierarchy of the entries indicates relative importance (top entries are mandatory) and progress through the requirements phase (top entries are available at the start of the requirements phase; bottom entries are supplied at the end of the requirements phase or at the start of the design phase). The reorganized sequence should also be reflected in the sequence of prompts in the edit/create functions of the MEDL-R system.

The list of entries in Table 3-4 is divided into seven groups. This grouping should be reflected in the sections of any revised Formatted Requirement Statement (FRS) report. Each group described below is composed of entries with a common theme:

- Group 1 contains entries that are either essential to the MEDL-R system concept or good requirement specifications in general.
- Group 2 contains entries that refine, supplement, or expand the information in Group 1.
- Group 3 entries require user judgment to characterize the requirement.
- Group 4 contains entries that show the decomposition of the system into components or the refinement of one requirement into consequent or dependent requirements.

ORIGINAL PAGE IS
OF POOR QUALITY

Table 3-4. Recommended Requirement Entries

GROUP	ENTRY	TYPE	LIST[a]	MEANING
1	IDENTIFICATION	STRING		NAME TAG OF THIS REQUIREMENT
1	DESCRIPTION	TEXT		ENGLISH-LANGUAGE EXPRESSION OF THIS REQUIREMENT
1	SOURCE	STRING		DOCUMENT REFERENCE
1	TEST-CRITERIA	TEXT		BENCHMARK DESCRPTION
2	EXPLANATION	TEXT		MISCELLANEOUS INFORMATION
2	SUBJECT	STRING	L	USER CATEGORY
2	CONSTRAINT	STRING		QUANTITY
3	NATURE	KEYWORD	L	CATEGORY
3	METRIC	KEYWORD	L	SOFTWARE QUALITY GOAL
3	MOTIVATION	KEYWORD	L	REASON FOR THIS REQUIREMENT
4	SCOPE	KEYWORD		RANGE OF INFLUENCE
4	SUBSYSTEM	STRING		PART OF SYSTEM
4	DERIVED-FROM	STRING	L	ORIGINATING REQUIREMENT
5	STATUS	KEYWORD		CURRENT STANDING
5	REPLACES	STRING	L	OVERRIDDEN REQUIREMENT
6	ORIGINATOR	STRING		NAME OF PERSON, GROUP
6	DEVELOPER	STRING		NAME OF PERSON, GROUP
6	REVIEWER	STRING		NAME OF PERSON, GROUP
7	FUNCTION-RESOLUTION	STRING	L	MODULE NAME
7	DATA-RESOLUTION	STRING	L	DATA SET NAME
7	RESOURCE-RESOLUTION	STRING	L	SPECIFIC HARDWARE

7282(80)

[a]AN L IN THIS COLUMN INDICATES THAT ONE OR MORE STRINGS OR KEYWORDS MAY BE PRESENT IN THE ASSOCIATED ENTRY.

- Group 5 contains entries that show the current standing of the requirement.
- Group 6 is devoted to naming the individuals or organizations assigned to each role in requirements analysis.
- Group 7 entries link the requirement with the design.

Sections 3.2.1 through 3.2.21 present either the recommendations for enhancement of an entry retained from the current MEDL-R system requirement or the reasons for including proposed new entries. Section 3.2.22 through 3.2.24 present the justification for deleting three of the current entries (VERSION, REPLACED-BY, and DERIVES).

3.2.1 IDENTIFICATION

The MEDL-R system should always prompt the user to supply the entry. The weaknesses of the current default sequence (R-1, R-2, and so forth) are inherent and would be present in any other default sequence.

By forcing the user to name each requirement, the MEDL-R system could help to organize the user's visualization of the structure of the requirements data base. Sample naming conventions provided by the user are mnemonics that indicate the requirement contents, original document section and page number, or any other categorizing scheme.

3.2.2 DESCRIPTION

No changes to the DESCRIPTION entry are recommended.

3.2.3 SOURCE

The SOURCE entry should be mandatory to ensure the retention of a link with an authoritative reference. The use of this entry will assist the reduction of the volume of material actually entered into the MEDL-R data base. The

ORIGINAL PAGE IS
OF POOR QUALITY

resolution of a detected inconsistency, ambiguity, or contradiction will be simplified by the inclusion of this entry.

3.2.4 TEST-CRITERIA

The TEST-CRITERIA entry is a recommended addition to the MEDL-R system requirement. The TEST-CRITERIA entry would be a text entry containing a procedure, benchmark, or other method for validating the requirement's presence in the final system.

The TEST-CRITERIA entry, if present, would help ensure that the DESCRIPTION entry is a precise and unambiguous statement.

3.2.5 EXPLANATION

No changes to the EXPLANATION entry are recommended.

3.2.6 SUBJECT

Each SUBJECT string entered by the user should be checked against a master list of previously entered SUBJECT strings of other requirements. If the string is already part of the list, the MEDL-R system will assume that the user wishes to connect the new requirement with others containing the string. If the string is not found, the entry is possibly a misspelling of a previous string. The MEDL-R system should notify the user that the SUBJECT about to be entered is new and allow the user to correct the spelling if an old string was the intended entry.

3.2.7 CONSTRAINT

The specified format of the CONSTRAINT string entry should be relaxed to allow the user to enter CONSTRAINTs that are not expressed only as a rate. This recommendation is contingent upon a finding that the MEDL-D system does not require CONSTRAINT to be expressed as a rate.

3.2.8 NATURE

The NATURE entry should be enhanced by the following two changes. First, the existing NATURE keyword list should be modified according to the list shown in Figure 3-2. This enhanced list takes into account the complete software life cycle and contains some aspects not referenced in the current list (maintenance, documentation, and testing). Second, the MEDL-R system should prompt the user for NATURE keywords in a way that takes advantage of the hierarchical structure of Figure 3-2. This may be done through menu prompts. This method of specifying the keywords has two advantages: (1) the hierarchical structure is ensured, and (2) the user does not have to memorize (or consult) the user's manual when entering NATURE keywords. For example, if the requirement is "to produce a specific document," the user/MEDL-R system interaction might be

```
MEDL-R > (NATURE)
         1 DEVELOPMENT
         2 PRODUCT
         3 POST-PRODUCT

USER   > 2

MEDL-R > (PRODUCT)
         1 OVERALL
         2 INTERNAL
         3 EXTERNAL
         4 TARGET-FACILITY
         5 DOCUMENTATION
         6 TESTING

USER   > 5
```

- DEVELOPMENT
  - OVERALL
  - TECHNIQUES
    - OVERALL
    - TOOLS
    - METHODOLOGY
  - MANAGEMENT
    - OVERALL
    - PROGRESS-REPORTING
    - MANPOWER
    - BUDGET
    - SCHEDULES
    - DELIVERABLES
  - DEVELOPMENT-FACILITY
    - OVERALL
    - HARDWARE
    - OPERATING-SYSTEM
    - LANGUAGES
  - PROJECTED-EVOLUTION
    - OVERALL
    - ENHANCEMENTS
    - FACILITY-CHANGES
    - PROVISIONS-FOR-EXPANSION
- PRODUCT
  - OVERALL
  - INTERNAL
    - OVERALL
    - PROCEDURAL
    - STRUCTURAL
    - TEMPORAL
    - DATA
    - INTERFACE
  - EXTERNAL
    - OVERALL
    - OPERATIONAL
    - PERFORMANCE
    - USER-INTERFACE
  - TARGET-FACILITY
    - OVERALL
    - HARDWARE
    - OPERATING-SYSTEM
    - LANGUAGES
    - REUSABLE-EXISTING-SOFTWARE
  - DOCUMENTATION
    - OVERALL
    - WORKING-TITLE
    - TOPICS-COVERED
    - DELIVERY-MEDIUM
    - NONSTANDARD-REQUESTS
  - TESTING
    - OVERALL
    - BENCHMARKS
    - CRITICAL-ERROR-DETECTION-RECOVERY
- POST-PRODUCT
  - OVERALL
  - INSTALLATION
    - OVERALL
    - STAGES
    - VALIDATION
    - TRAINING
  - MAINTENANCE
    - OVERALL
    - PROBLEM-REPORTING
    - LIABILITY
  - SUPPORT
    - OVERALL
    - CONSULTATION
    - NEW-RELEASES

Figure 3-2. Recommended NATURE Keywords

```
MEDL-R > (DOCUMENTATION)
         1 OVERALL
         2 WORKING-TITLE
         3 TOPICS-COVERED
         4 DELIVERY-MEDIUM
         5 NON-STANDARD-REQUESTS
USER   > 2
```

The keyword OVERALL found in the prompt indicates that the requirement cannot be subcategorized further and that additional prompts are not required.

This enhancement leaves the NATURE entry open-ended, and the user need not memorize all keyword spellings.

3.2.9 METRIC

The proposed METRIC requirement entry is a keyword type. The specific keywords and their definitions, taken from Reference 3, are presented in Table 3-5.

This entry will provide a "design to" criterion. Each set of requirements may be satisfied by any one of a large number of potential systems. If the requirements identify guidelines for applicable software quality factors, the number of potential solution systems is reduced because "nonquality" systems are excluded.

3.2.10 MOTIVATION

The current RESULTING-FROM entry should be changed to a MOTIVATION entry. The current name is easily confused with another MEDL-R system entry, DERIVED-FROM. No recommendations are made for changing the current keyword list.

3.2.11 SCOPE

The implicit relationship between the SCOPE entry and the SUBSYSTEM entry should be implemented.

Table 3-5. METRIC Keyword Definitions

KEYWORD	DEFINITION
INTEGRITY	SYSTEM'S CONTROL OVER UNAUTHORIZED ACCESS TO DATA OR SOFTWARE IN THE SYSTEM
USABILITY	EASE WITH WHICH THE SYSTEM CAN BE LEARNED OR OPERATED OR THE EASE WITH WHICH INPUT DATA ARE PREPARED OR OUTPUT DATA ARE INTERPRETED
CORRECTNESS	EXTENT TO WHICH THE SYSTEM SATISFIES ITS SPECIFICATIONS OR FULFILLS THE MISSION OBJECTIVES
RELIABILITY	EXTENT TO WHICH THE SYSTEM IS EXPECTED TO PERFORM ITS INTENDED FUNCTION WITH PRECISION AT ANY TIME
MAINTAINABILITY	TARGET LEVEL OF EFFORT NEEDED TO LOCATE AND REPAIR SYSTEM FAULTS ONCE THE SYSTEM IS OPERATIONAL
TESTABILITY	EASE WITH WHICH THE SYSTEM CAN BE TESTED TO ENSURE THAT IT PERFORMS ITS INTENDED FUNCTION
FLEXIBILITY	EASE WITH WHICH THE OPERATIONAL SYSTEM CAN BE MODIFIED
REUSEABILITY	EXTENT TO WHICH THE SYSTEM CAN BE USED IN ANOTHER APPLICATION
PORTABILITY	EASE WITH WHICH THE SYSTEM CAN BE TRANSFERRED FROM ONE HARDWARE CONFIGURATION AND/OR SOFTWARE ENVIRONMENT TO ANOTHER
INTEROPERABILITY	EASE WITH WHICH THE SYSTEM CAN BE COUPLED TO ANOTHER SYSTEM
EFFICIENCY	AMOUNT OF COMPUTING RESOURCES AND CODE ALLOCATED AND USED BY THE SYSTEM TO PERFORM ITS FUNCTION

7282/80

NOTE: THE INCLUSION OF ONE OR MORE OF THESE KEYWORDS UNDER THE METRIC ENTRY INDICATES THAT THE REQUIREMENT ADDRESSES THAT ASPECT OF THE SYSTEM.

If the SCOPE keyword LIMITED is entered, the SUBSYSTEM entry should become mandatory. The SUBSYSTEM names then would definitively describe the limits of the requirement.

Conversely, if a SUBSYSTEM entry is made, the MEDL-R system should insert the keyword LIMITED for SCOPE.

3.2.12 SUBSYSTEM

The recommended enhancement to the SUBSYSTEM entry is described in Section 3.2.11.

3.2.13 DERIVED-FROM

The MEDL-R system should be modified to compare the requirement IDENTIFICATION(s) entered under the DERIVED-FROM entry with the list of requirement IDENTIFICATION(s) present in the data base. If the DERIVED-FROM entry does not name a requirement in the data base, the MEDL-R system should give the user the opportunity to change the entry. The system should issue a warning if the DERIVED-FROM entry names a requirement whose STATUS is OBSOLETE.

3.2.14 STATUS

The current set of STATUS keywords (ACTIVE, SOFT, and OBSOLETE) does not completely reflect the evolution of a requirement. To obtain a better understanding of the evolution of a requirement, use of the keywords specified in Table 3-6 is recommended.

Figure 3-3 demonstrates the evolution of a requirement as it is reflected in Table 3-6.

3.2.15 REPLACES

The MEDL-R system should be modified to compare the requirement IDENTIFICATION(s) entered under the REPLACES entry with the list of requirement IDENTIFICATION(s) present in the data base. If the REPLACES entry does not name a requirement in the data base, the MEDL-R system should give the user the

Table 3-6. Recommended STATUS Keywords

KEYWORD	DEFINITION
TENTATIVE	A NEW REQUIREMENT ABOUT WHICH NOT MUCH IS KNOWN. IT IS EXPECTED TO PROGRESS TO ACTIVE, BUT IT MAY BECOME OBSOLETE. TENTATIVE IS THE DEFAULT FOR STATUS
ACTIVE	A REQUIREMENT (WHICH MAY BE NEW) ABOUT WHICH THE MAJORITY OF MEDL-R INFORMATION IS PRESENT. IT IS EXPECTED TO PROGRESS TO REVIEWED, BUT IT MAY BECOME QUESTIONED
REVIEWED	A REQUIREMENT ABOUT WHICH "EVERYTHING" IS KNOWN. IT HAS BEEN VALIDATED BY AN INDEPENDENT REVIEWER WHO HAS CONSIDERED IT TO BE COMPLETE AND CORRECTLY RELATED TO OTHER REQUIREMENTS. IT MAY BECOME OBSOLETE DUE TO THE DISCOVERY OF AN ERROR OR THROUGH A CHANGE IN THE ORIGINATOR'S SPECIFICATIONS, BUT THERE IS A VERY GOOD CHANCE THAT IT WILL BECOME PART OF THE FINAL SYSTEM
QUESTIONED	A REQUIREMENT THAT IS DISPUTED. DOUBT EXISTS ABOUT THE APPLICABILITY, FEASIBILITY, AND SO FORTH, OF THIS REQUIREMENT, AND WHETHER IT WILL BE INCLUDED IN THE FINAL SPECIFICATIONS WITHOUT MODIFICATION IS UNCERTAIN. A QUESTIONED REQUIREMENT MAY BECOME EITHER OBSOLETE OR ACTIVE, DEPENDING UPON THE RESOLUTION OF THE DISPUTE
OBSOLETE	A REQUIREMENT THAT FOR ANY REASON HAS BEEN MARKED FOR NO FURTHER CONSIDERATION. OBSOLETE REQUIREMENTS ARE RETAINED AS REMINDERS OF "BLIND ALLEYS" OR ERRORS AND MAY NOT BE CHANGED IN ANY WAY

7282/180

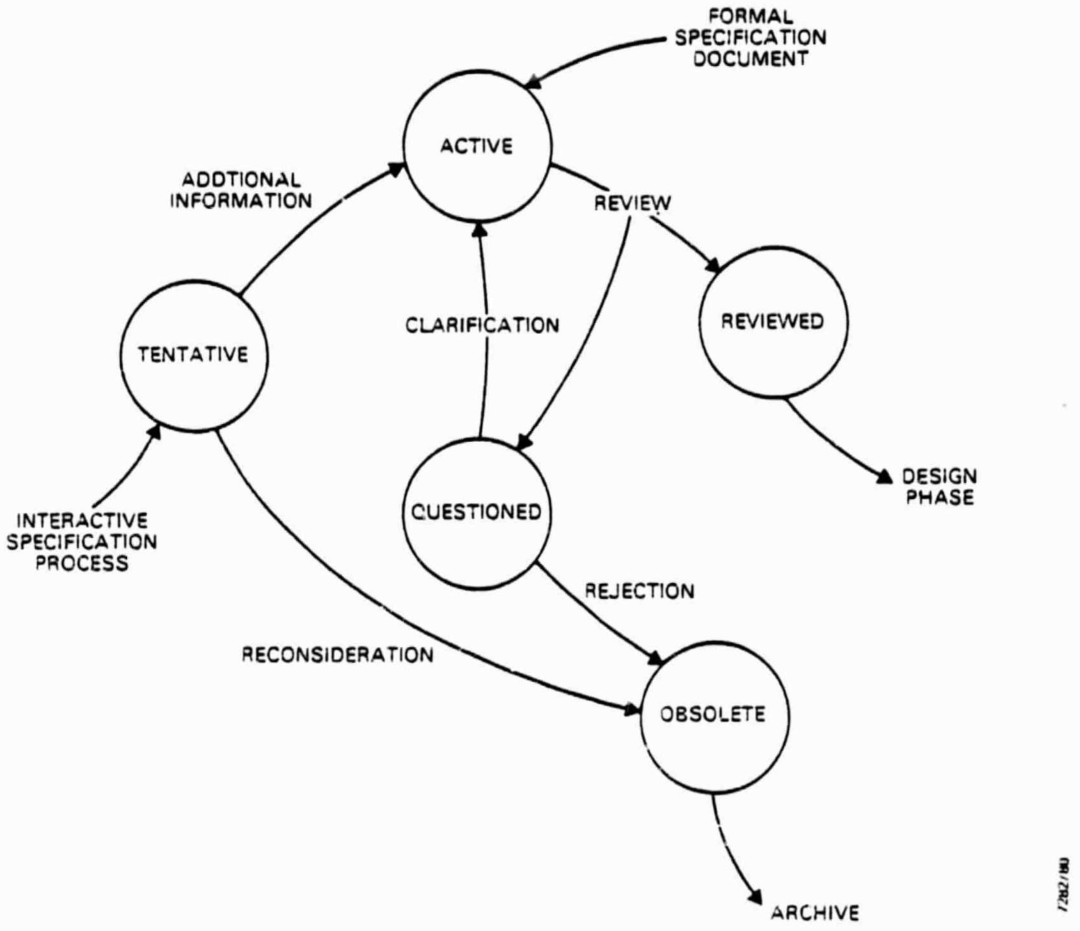

Figure 3-3. Requirement Evolution as Reflected by STATUS Keyword

ORIGINAL PAGE IS
OF POOR QUALITY

opportunity to change the entry. The system should issue a warning if the REPLACES entry names a requirement whose STATUS is OBSOLETE.

3.2.16 ORIGINATOR

No changes to the ORIGINATOR entry are recommended.

3.2.17 DEVELOPER

The name of the current RESPONSIBILITY entry should be changed to DEVELOPER. The current name might be confused with the proposed new entry REVIEWER, which names a person or group with a particular kind of responsibility.

3.2.18 REVIEWER

The REVIEWER entry should be added to the MEDL-R system requirement. This entry is intended to include the name of a person (or group) who has the responsibility of certifying that the requirement is complete and correctly related to other requirements. This person (or group) is responsible for recommending a change in the STATUS entry from ACTIVE to either QUESTIONED or REVIEWED (see Section 3.2.14).

3.2.19 FUNCTIONAL-RESOLUTION

No changes to the FUNCTIONAL-RESOLUTION entry are recommended.

3.2.20 DATA-RESOLUTION

No changes to the DATA-RESOLUTION entry are recommended.

3.2.21 RESOURCE-RESOLUTION

No changes to the RESOURCE-RESOLUTION entry are recommended.

3.2.22 VERSION

The VERSION entry should not be used for individual requirements; instead, the system should perform this housekeeping task. The system version name can be expected to remain constant over several MEDL-R sessions (at least). The current operation of manually specifying the version for each requirement can lead to erroneous entries.

Because the version name is a valid component of each requirement when the requirement is considered outside the data base, user control is still necessary. It would be better to specify a version name at the start of each session and to allow the MEDL-R system to attach the name to each requirement modified or created during the session.

3.2.23 REPLACED-BY

The REPLACED-BY entry should not be used for individual requirements. The MEDL-R system should be modified to resolve a REPLACES entry with an automatic REPLACED-BY entry under the appropriate requirement. This modification would relieve the user of a complicated, error-prone bookkeeping task.

This modification has many ramifications that would affect the STATUS and DERIVED-FROM entries. The full extent of the modifications is estimated to be quite large; however, the benefits of automating this task would be realized in a reduction in errors and the development of a systematic and reliable tracing and auditing method.

3.2.24 DERIVES

The DERIVES entry should not be used for individual requirements. The MEDL-R system should be modified to resolve a DERIVED-FROM entry with an automatic DERIVES entry under the appropriate requirements.

As in the case of the REPLACED-BY entry, this deletion would require extensive planning and design prior to implementation. The benefits realized from this modification would be similar to those mentioned in Section 3.2.23.

SECTION 4 - EVALUATION SUMMARY

This section summarizes the results of the MEDL-R system evaluation and proposes a sequence of implementation for the recommended modifications to the system.

4.1 EVALUATION AND RECOMMENDATIONS

The evaluation and recommendations are based on experience in processing small numbers of requirements with the MEDL-R system. The current version of the MEDL-R system tends to discourage the user from entering large numbers of requirements because of inadequacies in its implementation and difficulty in interpreting the final results. Thus, the evaluation and recommendations are directed toward improving the MEDL-R system from an operational standpoint, with the hope that an improved system will attract a larger user community. The validity of the MEDL-R system concept of requirements analysis can be tested only after a number of users are familiar with it in varied applications, at which point enhancements to the analysis performed might also be attempted.

The major findings resulting from the evaluation of the MEDL-R system are as follows:

- The MEDL-R system represents a good starting point in the development of a new concept of Requirements Analysis Languages (RALs).
- The MEDL-R system contains the majority of functions desirable in the Code 580 environment.
- The MEDL-R system adheres to an interactive approach (rather than a batch approach) to RALs.
- The majority of the components of the MEDL-R language syntax are pertinent and useful to Code 580.
- The current structure of the system can be modified into a more operational implementation.

The recommended changes to the MEDL-R system are specifically directed toward correcting some awkwardnesses in the details of implementation and improving the MEDL-R system operationally. The most important change to be made to MEDL-R would be reorganization of the system as outlined in Section 2.4. The most extensive portion of this effort would be the consolidation of the two current edit subsystems (Create and Update) into one subsystem. The single edit subsystem would isolate direct user contact with the requirements into one portion of the system and thus reduce the programming effort required by further enhancement, extension, or modification to the language syntax. One subsystem would also reduce the user's learning time, because only one edit procedure would need to be learned. The reorganization effort would also be directed toward eliminating superfluous external files and allowing better online reporting capability.

Following the reorganization of the MEDL-R system, the language syntax modifications would be implemented. The recommended modifications are classified below:

- New language entries--TEST-CRITERIA (Section 3.2.4), METRIC (Section 3.2.9), and REVIEWER (Section 3.2.18)
- Default values for entries--SCOPE (Section 3.2.11), STATUS (Section 3.2.14), and VERSION (Section 3.2.22); the recommended enhancements would improve overall configuration control
- Revised keywords (and their presentation in a menu format)--NATURE (Section 3.2.8) and STATUS (Section 3.2.14)
- Automatic program-supplied values for some entries based on the current value of others--SCOPE and SUBSYSTEM (Section 3.2.11), DERIVED-FROM and DERIVES (Section 3.2.24), and REPLACES and REPLACED-BY

(Section 3.2.23); the recommended enhancement would represent an automatic linking feature to ensure the credibility of the data base and allow additional analysis

Other recommended modifications to the MEDL-R system such as (1) extending the QUERY syntax to a more general form covering a larger subset of the MEDL-R entries (Section 2.1.4) and (2) expanding the content of the Analyzer subsystem SUMMARY report (Section 2.1.5) could also be implemented after the reorganization of the MEDL-R system structure. These changes represent the only proposed improvements to the analysis portion of the MEDL-R system. The proposed new functions of combining MEDL-R requirement data bases and creating a subset data base (Section 2.1.7) would be implemented (as all new analysis features should be) only after more experience is gained with large requirement systems.

4.2 FUTURE PLANS FOR MEDL-R IN CODE 580 ENVIRONMENT

Code 580 proposes to conduct a pilot test of the MEDL-R system as it is currently installed on the PDP-11/70 in the Software Engineering Laboratory (SEL). Several development tasks currently being monitored by the SEL will use MEDL-R as part of the requirement specification analysis phase. A senior analyst with approximately 3 to 4 years' experience in Code 580 software development will extract the functional specifications from the requirements document. The specifications will be translated to the MEDL-R language, and an attempt will be made to link the specifications to actual program modules, data sets, or system resources. Other features of MEDL-R will be tested at the discretion of the analyst. The results of the pilot tests should be available in spring 1981.

REFERENCES

1. Martin Marietta Aerospace, Multi-Level Expression Design System Requirements Level Description Manual, P. Scheffer and A. Musser, February 1979

2. --, MEDL-R Language User's Guide, P. Scheffer and A. Musser, February 1979

3. Rome Air Development Center, Factors in Software Quality, Volume III, RADC-TR-77-369, Preliminary Handbook on Software Quality for an Acquisition Manager, J. McCall, P. Richards, and G. Walters, November 1977

BIBLIOGRAPHY OF SEL LITERATURE

Anderson, L., "SEL Library Software User's Guide," Computer Sciences-Technicolor Associates, Technical Memorandum, June 1980

Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development for Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Banks, F. K., "Configuration Analysis Tool (CAT) Design," Computer Sciences Corporation, Technical Memorandum, March 1980

Basili, V. R., "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., "SEL Relationships for Programming Measurement and Estimation," University of Maryland, Technical Memorandum, October 1980

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: Computer Societies Press, 1980 (also designated SEL-80-008)

Basili, V. R., and J. Beane, "Can the Parr Curve Help with the Manpower Distribution and Resource Estimation Problems?", Journal of Systems and Software, February 1981, vol. 2, no. 1

Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

Basili, V. R., and T. Phillips, "Validating Metrics on Project Data," University of Maryland, Technical Memorandum, December 1981

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity and Cost, October 1979

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

Basili, V. R., and M. V. Zelkowitz, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977

Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: Computer Societies Press, 1978

Church, V. E., "User's Guides for SEL PDP-11/70 Programs," Computer Sciences Corporation, Technical Memorandum. March 1980

Freburger, K., "A Model of the Software Life Cycle" (paper prepared for the University of Maryland, December 1978)

Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977 (also designated SEL-77-005)

Hislop, G., "Some Tests of Halstead Measures" (paper prepared for the University of Maryland, December 1978)

Lange, S. F., "A Child's Garden of Complexity Measures" (paper prepared for the University of Maryland, December 1978)

Mapp, T. E., "Applicability of the Rayleigh Curve to the SEL Environment" (paper prepared for the University of Maryland, December 1978)

Miller, A. M., "A Survey of Several Reliability Models" (paper prepared for the University of Maryland, December 1978)

National Aeronautics and Space Administration (NASA), NASA Software Research Technology Workshop (proceedings), March 1980

Page, G., "Software Engineering Course Evaluation," Computer Sciences Corporation, Technical Memorandum, December 1977

Parr, F., and D. Weiss, "Concepts Used in the Change Report Form," NASA, Goddard Space Flight Center, Technical Memorandum, May 1978

Perricone, B. T., "Relationships Between Computer Software and Associated Errors: Empirical Investigation" (paper prepared for the University of Maryland, December 1981)

Reiter, R. W., "The Nature, Organization, Measurement, and Management of Software Complexity" (paper prepared for the University of Maryland, December 1976)

Scheffer, P. A., and C. E. Velez, "GSFC NAVPAK Design Higher Order Languages Study: Addendum," Martin Marietta Corporation, Technical Memorandum, September 1977

Software Engineering Laboratory, SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

--, SEL-77-001, The Software Engineering Laboratory, V. R. Basili, M. V. Zelkowitz, F. E. McGarry, et al., May 1977

--, SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

--, SEL-77-003, Structured FORTRAN Preprocessor (SFORT), B. Chu, D. S. Wilson, and R. Beard, September 1977

--, SEL-77-004, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

--, SEL-78-001, FORTRAN Static Source Code Analyzer (SAP) Design and Module Descriptions, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, January 1978

--, SEL-78-002, FORTRAN Static Source Code Analyzer (SAP) User's Guide, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

--, SEL-78-003, Evaluation of Draper NAVPAK Software Design, K. Tasaki and F. E. McGarry, June 1978

--, SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson, B. Chu, and G. Page, September 1978

--, SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

--, SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer, November 1978

--, SEL-79-001, SIMPL-D Data Base Reference Manual, M. V. Zelkowitz, July 1979

--, SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

--, SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, S. R. Waligora, and A. L. Green, August 1979

--, SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and F. E. McGarry, September 1979

--, SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

--, SEL-80-001, Configuration Analysis Tool (CAT) Functional Requirements/Specifications, F. K. Banks, C. E. Goorevich, and A. L. Green, February 1980

--, SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker, C. E. Goorevich, and A. L. Green, May 1980

--, SEL-80-003, Multimission Modular Spacecraft Ground Support System (MSS/GSSS) State-of-the-Art Computer System/Compatibility Study, T. Weldon, M. McClellan, P. Liebertz, et al., May 1980

--, SEL-80-004, System Description and User's Guide for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, W. J. Decker, J. G. Garrahan, et al., October 1980

--, SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

--, SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

--, SEL-80-007, _An Appraisal of Selected Cost/Resource Estimation Models for Software Systems_, J. F. Cook and F. E. McGarry, December 1980

--, SEL-81-001, _Guide to Data Collection_, V. E. Church, D. N. Card, F. E. McGarry, et al., September 1981

--, SEL-81-002, _Software Engineering Laboratory (SEL) Data Base Organization and User's Guide_, D. C. Wyckoff, G. Page, F. E. McGarry, et al., September 1981

--, SEL-81-003, _Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description_, D. N. Card, D. C. Wyckoff, G. Page, et al., September 1981

--, SEL-81-004, _The Software Engineering Laboratory_, D. N. Card, F. E. McGarry, G. Page, et al., September 1981

--, SEL-81-005, _Standard Approach to Software Development_, V. E. Church, F. E. McGarry, G. Page, et al., September 1981

--, SEL-81-006, _Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide_, W. Taylor and W. J. Decker, December 1981

--, SEL-81-007, _Software Engineering Laboratory (SEL) Compendium of Tools_, W. J. Decker, E. J. Smith, A. L. Green, et al., February 1981

--, SEL-81-008, _Cost and Reliability Estimating Models (CAREM) User's Guide_, J. F. Cook and E. Edwards, February 1981

--, SEL-81-009, _Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation_, W. J. Decker, A. L. Green, and F. E. McGarry, March 1981

--, SEL-81-010, _Performance and Evaluation of Independent Software Verification and Integration Process_, G. Page and F. E. McGarry, May 1981

--, SEL-81-011, _Evaluating Software Development by Analysis of Change Data_, D. M. Weiss, November 1981

--, SEL-81-012, _Software Engineering Laboratory_, G. O. Picasso, December 1981

--, SEL-81-013, _Proceedings From the Sixth Annual Software Engineering Workshop_, December 1981

--, SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

Turner, C., G. Caron, and G. Brement, "NASA/SEL Data Compendium," Data and Analysis Center for Software, Special Publication, April 1981

Turner, C., and G. Caron, "A Comparison of RADC and NASA/SEL Software Development Data," Data and Analysis Center for Software, Special Publication, May 1981

Weiss, D. M., "Error and Change Analysis," Naval Research Laboratory, Technical Memorandum, December 1977

Williamson, I. M., "Resource Model Testing and Information," Naval Research Laboratory, Technical Memorandum, July 1979

Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: Computer Societies Press, 1979

Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981